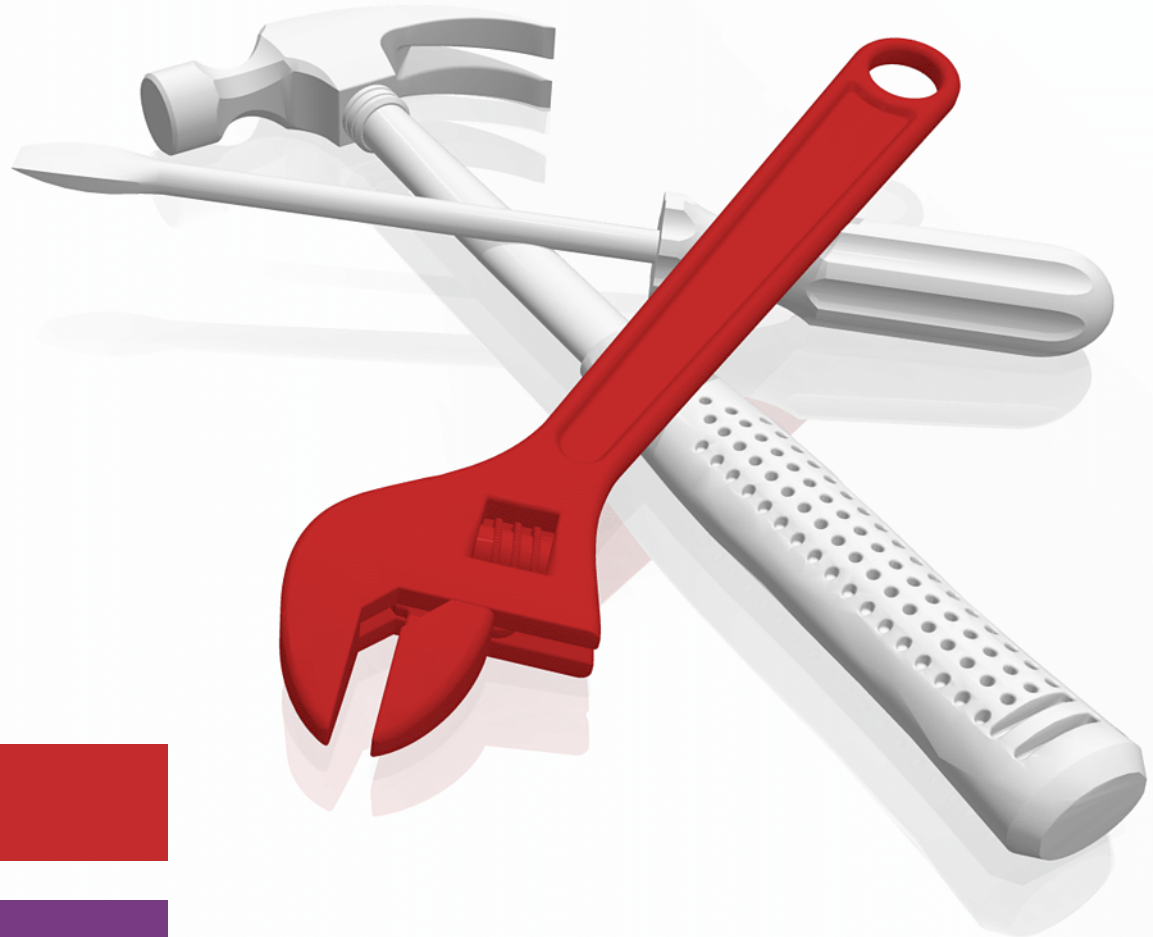


# Using Ansible for Automation in IBM Power Environments

The IBM Redbooks Team



 Cloud

Power Systems





IBM Redbooks

**Using Ansible for Automation in IBM Power Environments**

November 2024

**Note:** Before using this information and the product it supports, read the information in “Notices” on page vii.

### **First Edition (November 2024)**

This edition applies to the following product versions:

Ansible Core 2.14

Ansible 2.10.8

Ansible extension 2.6.92 and 2.7.98

Red Hat Ansible Automation Platform 2.4-1.2 (ppc64le)

AIX 7.2 TL5 and AIX 7.3 TL1

IBM i 7.3 TR13, IBM i 7.4 TR7, and IBM i 7.5

IBM i Modernization Engine for Lifecycle Integration (Merlin) 1.0

Red Hat Enterprise Linux Server (RHEL) 7.9 (ppc64 - Big Endian)

Red Hat Enterprise Linux Server 8.4 (ppc64le)

Red Hat Enterprise Linux Server 9.2 (ppc64le)

SUSE Linux Enterprise Server 15 SP 5 (ppc64le)

Ubuntu 20.04.6 (ppc64le)

Ubuntu 22.04.3 (ppc64le)

VisualStudio Code 1.83.0

IBM PowerVM Virtual I/O Server (VIOS) 3.1.3 and Virtual I/O Server 3.1.4

IBM PowerVM Virtual I/O Server 4.1.0.10

IBM Hardware Management Console (HMC) 8.8.7, 9.1, or later

VisualStudio Code 1.83.0

© Copyright International Business Machines Corporation 2024. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	vii
Trademarks .....	viii
<b>Preface</b> .....	ix
Authors .....	ix
Now you can become a published author, too! .....	xiii
Comments welcome .....	xiii
Stay connected to IBM Redbooks .....	xiv
<b>Chapter 1. Introducing Ansible and IBM Power</b> .....	1
1.1 Why automation .....	2
1.1.1 Orchestration versus automation .....	3
1.2 Automation tools and techniques .....	4
1.2.1 Common IT automation tools .....	5
1.3 Understanding Ansible: A powerful automation tool .....	5
1.3.1 Ansible architecture .....	6
1.3.2 Options for implementing Ansible .....	7
1.3.3 Ansible Automation Platform .....	10
1.3.4 Event-driven automation .....	15
1.3.5 Infrastructure as Code: integration of Ansible and Terraform .....	17
1.3.6 Provisioning .....	19
1.3.7 Patch management .....	20
1.3.8 Security and compliance .....	20
1.3.9 Configuration management .....	22
1.3.10 Business continuity .....	23
1.3.11 Application development .....	23
1.4 Introducing IBM Power .....	24
1.4.1 IBM Power high availability .....	25
1.4.2 IBM Power security .....	25
1.4.3 IBM Power, operational efficiency, and sustainability .....	26
1.4.4 Streamlining AI operations with advanced on-chip technologies .....	26
1.4.5 POWER processors and architecture .....	26
1.4.6 PowerVM and virtualization .....	28
1.4.7 Supported operating systems .....	29
1.4.8 Key benefits of IBM Power compared to x86 servers .....	29
1.5 Ansible for Power .....	30
1.5.1 IBM Power Collections on Ansible Galaxy .....	30
1.5.2 IBM Power Collections on Red Hat Automation Hub .....	31
1.5.3 Ansible for Linux on Power .....	31
1.5.4 Ansible for AIX .....	40
1.5.5 Ansible for IBM i .....	44
1.5.6 Ansible for IBM Power Hardware Management Console .....	52
1.5.7 Ansible for Power Virtual I/O server .....	53
1.5.8 Ansible for IBM Power Systems Virtual Server .....	54
1.5.9 Ansible for applications .....	56
<b>Chapter 2. Ansible architecture and design</b> .....	61
2.1 Ansible architecture and components .....	62
2.1.1 Controller and client functions .....	63

2.2	Understanding the Ansible declarative language	64
2.2.1	YAML structure	64
2.2.2	Jinja2	67
2.3	Understanding an Ansible inventory	68
2.3.1	Overview of an Ansible inventory	68
2.3.2	Overview of dynamic inventory	70
2.4	Ansible tasks, playbooks, and modules	79
2.4.1	Creating Ansible playbooks	79
2.5	Ansible roles and collections	86
2.5.1	Understanding roles in Ansible	86
2.5.2	Creating and structuring Ansible roles	87
2.5.3	Sharing and reusing roles in multiple playbooks	89
2.5.4	Role dependencies and role-based variables	90
2.5.5	Using collections	91
2.6	Best practices for playbook and role design	92
2.6.1	Writing modular and reusable playbooks	93
2.6.2	Using Ansible Galaxy for role management	94
2.7	Creating versions and documenting playbooks and roles	96
2.7.1	Creating versions of playbooks and roles	96
2.7.2	Common scenarios when using Git with Ansible	96
2.8	Testing and validating playbooks and roles	100
2.8.1	Testing playbooks and roles	100
2.8.2	Validating playbooks and roles	101
<b>Chapter 3</b>	<b>Getting started with Ansible</b>	<b>103</b>
3.1	Designing your Ansible environment	104
3.1.1	Starting simple: Ansible Core and Ansible Community	104
3.1.2	Scaling up: Ansible Automation Platform	105
3.1.3	Enterprise-ready environment	118
3.1.4	Developing an “automation first” attitude	120
3.2	Choosing the Ansible Controller node	121
3.3	Installing your Ansible control node	121
3.3.1	Linux as an Ansible Controller	122
3.3.2	AIX as an Ansible Controller	134
3.3.3	IBM i as an Ansible Controller	143
3.4	Preparing your systems to be Ansible clients	149
3.4.1	Linux as an Ansible managed client	149
3.4.2	AIX as an Ansible managed client	150
3.4.3	IBM i as an Ansible managed client	151
3.4.4	Virtual I/O Server as an Ansible managed client	160
3.4.5	Red Hat OpenShift as an Ansible managed client	169
3.4.6	IBM Power Hardware Management Console as an Ansible managed client	180
<b>Chapter 4</b>	<b>Automated application deployment on IBM Power servers</b>	<b>191</b>
4.1	Deploying and managing applications by using Ansible on Power servers	192
4.2	Automated application deployment on Power servers	192
4.2.1	Ansible content for IBM Power	192
4.2.2	IBM AIX, IBM i, and Linux on Power collections for Ansible	192
4.3	Deploying a simple Node.js application	193
4.4	Orchestrating multitier application deployments	194
4.4.1	Orchestration in the world of Kubernetes	194
4.5	Continuous integration and continuous deployment pipelines with Ansible	194
4.5.1	CI/CD when using Ansible for IBM i	195

4.6 Oracle DB automation on Power. . . . .	196
4.6.1 Why businesses opt for AIX to host their databases. . . . .	197
4.6.2 Automating the deployment of a single-node Oracle database with Ansible . . .	197
4.6.3 Automating the deployment of Oracle RAC with Ansible . . . . .	201
4.6.4 Automating Oracle DBA operations . . . . .	214
4.7 SAP automation . . . . .	222
4.7.1 Red Hat Enterprise Linux System Roles for SAP . . . . .	225
4.7.2 Using the SAP LinuxLab automation . . . . .	228
<b>Chapter 5. Infrastructure as Code by using Ansible . . . . .</b>	<b>239</b>
5.1 IBM Power Virtualization Center . . . . .	240
5.1.1 Advantages of PowerVC. . . . .	240
5.1.2 Using the OpenStack Cloud modules. . . . .	240
5.1.3 Using the URI modules to interact with PowerVC API services . . . . .	255
5.2 IBM Power Systems Virtual Server . . . . .	265
5.2.1 Using the IBM Cloud collection for Power Systems Virtual Server . . . . .	265
5.2.2 Using the URI module for Power Systems Virtual Server. . . . .	267
<b>Chapter 6. Day 2 management operations . . . . .</b>	<b>279</b>
6.1 Introducing Day 2 operations . . . . .	280
6.1.1 Storage . . . . .	280
6.1.2 Security and compliance. . . . .	281
6.1.3 Patches or upgrades. . . . .	281
6.1.4 Configuration and tuning. . . . .	282
6.2 Day 2 operations in Linux servers. . . . .	282
6.2.1 Installing system roles for Ansible automation . . . . .	282
6.2.2 Storage . . . . .	283
6.2.3 Security and compliance. . . . .	288
6.2.4 Patches and upgrades . . . . .	293
6.2.5 Configuration tuning . . . . .	297
6.3 Day 2 operations in AIX environments . . . . .	299
6.3.1 Storage . . . . .	299
6.3.2 Security. . . . .	311
6.3.3 Fixes . . . . .	317
6.3.4 Configuration tuning . . . . .	321
6.4 Day 2 operations in IBM i environments . . . . .	323
6.4.1 Storage . . . . .	323
6.4.2 Security and compliance. . . . .	325
6.4.3 Patch management. . . . .	327
6.4.4 Configuration tuning . . . . .	328
<b>Chapter 7. Future trends and directions . . . . .</b>	<b>331</b>
7.1 Ansible and IBM Power Roadmap . . . . .	332
7.1.1 Working closely with the IBM Power collections and their contents . . . . .	332
7.2 Roadmap for Ansible automation in the Power ecosystem. . . . .	333
7.2.1 Ansible Automation Platform on IBM Power. . . . .	334
7.2.2 Visual Studio Code . . . . .	334
7.2.3 IBM watsonx Code Assistant for Red Hat Ansible Lightspeed . . . . .	342
<b>Appendix A. Unveiling IBM i Modernization Engine for Lifecycle Integration . . . . .</b>	<b>349</b>
Introduction . . . . .	350
What is IBM i Merlin . . . . .	350
The role of IBM i Merlin in the IBM i market . . . . .	351
IBM i Merlin: Problem-solving capabilities. . . . .	352

Benefits of IBM i Merlin for IBM i modernization . . . . .	352
Decades of collaboration: IBM and ARCAD . . . . .	353
Components of IBM i Merlin . . . . .	353
Comprehensive overview of IBM i Merlin content. . . . .	355
Ansible integration for IBM i lifecycle management through IBM i Merlin . . . . .	358
The business demands for DevOps on IBM i . . . . .	362
IBM i Merlin for IBM i developers . . . . .	375
IBM i Merlin requirements . . . . .	386
<b>Abbreviations and acronyms . . . . .</b>	<b>389</b>
<b>Related publications . . . . .</b>	<b>391</b>
IBM Redbooks . . . . .	391
Help from IBM . . . . .	391



# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

# Trademarks

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <https://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

AIX®	IBM Sterling®	PowerVM®
Db2®	IBM Z®	QRadar®
DS8000®	Instana®	Rational®
IBM®	Integrated Language Environment®	Redbooks®
IBM Cloud®	Micro-Partitioning®	Redbooks (logo)  ®
IBM Cloud Pak®	Passport Advantage®	SoftLayer®
IBM Consulting™	POWER®	Sterling™
IBM FlashSystem®	Power8®	SystemMirror®
IBM Instana™	Power9®	Turbonomic®
IBM Security®	PowerHA®	WebSphere®

The following terms are trademarks of other companies:

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel Xeon, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

ITIL is a Registered Trade Mark of AXELOS Limited.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Red Hat, Ansible, OpenShift, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

# Preface

This IBM Redbooks® publication helps you install, tailor, and configure an automation environment by using Red Hat Ansible in an IBM Power server environment. Ansible is a versatile IT automation platform that you use to deploy and maintain your applications and systems. With Ansible, you can automate almost anything: code deployment, network configuration, server infrastructure deployment, security and patch management, and cloud management. Ansible is implemented in a human-readable language (YAML) and uses Secure Shell (SSH) to connect to the managed systems, with no agents to install on remote systems.

This IBM Redbooks publication shows you how to integrate Ansible to manage all aspects of your IBM Power infrastructure, including server hardware, the Hardware Management Console (HMC), IBM PowerVM®, IBM PowerVC, IBM AIX®, IBM i, and Linux on Power. We provide guidance about where to run your Ansible automation controller nodes, demonstrate how Ansible can be installed on any operating system (OS) that is supported on IBM Power, and show you how to set up your IBM Power infrastructure components to be managed by using Ansible.

This publication is intended for use by anyone who is interested in automaton by using Ansible, whether they are getting started or they are experts on Ansible and want to understand how to integrate IBM Power into their existing environment.

## Authors

This book was produced by a team of specialists from around the world with IBM Redbooks.

**Tim Simon** is an IBM Redbooks Project Leader who is based in Tulsa, Oklahoma, US. He has over 40 years of experience with IBM®, primarily in a technical sales role working with customers to help them create IBM solutions to solve their business problems. He holds a BS degree in Math from Towson University in Maryland. He has worked with many IBM products and has extensive experience creating customer solutions by using IBM Power, IBM Storage, and IBM Z® throughout his career.

**Jose Martin Abeleira** is a Senior Systems and Storage Administrator at DGI (a Uruguay Taxes Collection Agency). He is a Gold Redbooks Author, Certified Consulting IT Specialist, and IBM Certified Systems Expert Enterprise Technical Support for IBM AIX and Linux in Montevideo, Uruguay. He has worked with IBM for 8 years and has 18 years of AIX experience. He holds an Information Systems degree from Universidad Ort Uruguay. His areas of expertise include IBM Power, AIX, UNIX, Linux, Live Partition Mobility (LPM), IBM PowerHA® SystemMirror®, storage area network (SAN), and storage systems from IBM and other vendors. He teaches Systems Administration for the Systems Engineer career path at the Universidad Catolica del Uruguay, and Infrastructure Administration in the Computer Technologist career path that was created by the joint venture between Universidad del la Republica Uruguay, Universidad del Trabajo del Uruguay, and Universidad Tecnologica.

**Shahid Ali** is a Cloud Solution Lead for the MEA Region. At the time of writing, he is based in Riyadh, Saudi Arabia, and leading hybrid multi-cloud solutions in the MEA region. He is an experienced Enterprise Architect who joined IBM 5 years ago as an Enterprise Architect. He has 28 years of experience as an architect and consultant. Before joining IBM, he provided consultancy services for some of the largest projects in Saudi Arabia for the Ministries of Interior, Education, and Labor, and related organizations. These projects produced nationwide solutions for fingerprinting, country-wide secure networks, smart ID cards, e-services portals, enterprise resource planning systems, and massive, open online courses platforms. He has several IBM and industry certifications, and is also a member of the IBM Academy of Technology.

**Vijaybabu Anaimuthu** is a Technical Consultant at IBM Systems Experts Labs who is based in India. He holds a bachelor degree in Electrical and Electronics Engineering from Anna University, Chennai. He has over 15 years of experience working with customers designing and deploying solutions on IBM Power servers and AIX. He focuses on IT Infrastructure Enterprise Solutions, technical enablement and implementations relative to IBM Power servers, Enterprise Pools, performance, and automation. His areas of expertise include capacity planning, migration planning, system performance, and automation.

**Sambasiva Andaluri** (Sam) is an experienced developer turned Solution Architect Leader with over 30 years of experience. For the past decade, he has been a pre-sales and post-sales solution architect for trading systems at Fidessa, a pre-sales solution architect at AWS, and an Site Reliability Engineering onboarding independent software vendors (ISVs) for Google marketplace.

**Marcelo Avalos Del Carpio** is a Cloud Architect at Kyndryl Consulting who is based in Uruguay with over 9 years of experience in IT. A former IBM leader, he specialized in deploying IBM technical solutions for key accounts across South America and North America. He holds an Electronic Systems Engineering degree from Escuela Militar de Ingeniería, Bolivia, and a master degree in Project Management from GSPM UCI, Costa Rica. He is certified by The Open Group, and specializes in IT infrastructure, cloud platforms, and DevOps, drawing from frameworks such as PMI, ITIL, and TOGAF.

**Thomas Baumann** is Senior Systems Engineer and Managing Director of ACP IT Consulting GmbH (formerly tiri GmbH) who is based in Hamburg, Germany, which is an IBM Business Partner and a Red Hat Premier Partner. He has over 30 years of experience in computer technology, and is also a trainer for IBM Software, Ansible Automation, Linux and cloud, and Security and Threat Management.

**Ivaylo Bozhinov** is an IBM Power subject matter expert (SME) who is based at IBM Bulgaria. His main area of expertise is solving complex hardware and software issues on IBM Power products, IBM AIX, Virtual I/O Server (VIOS), HMC, IBM i, PowerVM, and Linux on Power servers. He has been with IBM since 2015, and provides reactive break-patch, proactive, preventive, and cognitive support.

**Carlo Castillo** is a Client Services Manager for IBM Power for Right Computer Systems, an IBM Business Partner and Red Hat partner who is based in the Philippines. He has over 32 years of experience in pre-sales and post-sales support; designing full IBM infrastructure solutions; creating pre-sales configurations; performing IBM Power installation, implementation, and integration services; providing post-sales services and technical support for customers; and conducting presentations at customer engagements and corporate events. He was the first IBM certified AIX Technical Support engineer in the Philippines in 1999. As training coordinator during his Right Computer Systems tenure as an IBM Authorized Training Provider from 2007 to 2014, he also administered the IBM Power curriculum, and conducted IBM training classes about AIX, PureSystems, PowerVM, and IBM i. He holds a degree in Computer Data Processing Management from the Polytechnic University of the Philippines.

**Rafael Cezario** is a Senior Solutions Engineer at Blue Trust, an IBM Business Partner who is based in Brazil. Previously, he was an employee of IBM, where he worked as a pre-sales technical resource on IBM Power servers. He has 19 years of IT experience, and has worked on various infrastructure projects, including design, implementation, demonstration, installation, and integration of solutions. He has worked with various software on the IBM Power platform, such as PowerVM implementations that include Shared Ethernet Adapter and virtual network interface card, PowerVC, PowerSC, Red Hat OpenShift, Ansible, and Network Installation Manager (NIM) server. During his career at IBM, he served as a consultant for large clients regarding IBM Power and AIX, performed pre-sales and post-sales activities, and performed presentations and demonstrations for clients. He has worked in several areas of infrastructure during his career and became certified in several of these technologies, such as Cisco CCNA, Nutanix NCA, and IBM AIX. He holds a degree in Electrical Engineering with a specialization in Telecommunications from the Instituto de Ensino Superior de Brasília (IESB).

**Stuart Cunliffe** is a solution engineer within IBM Technology Expert Labs who is based in the UK. He specializes in IBM Power servers. He has worked for IBM since graduating from Leeds Metropolitan University in 1995, and has held roles in IBM Demonstration Group, Global Technologies Services (GTS) System Outsourcing, eBusiness hosting, and IBM Technical Support. A key area of his expertise is helping customers design and deliver automation across their IBM Power environment with solutions that involve tools such as Red Hat Ansible, HashiCorp Terraform, and IBM PowerVC.

**Nilabja Haldar** is an experienced Cloud Architect and Site Reliability Engineer and a certified AWS, Google Cloud Platform, Azure, IBM Cloud®, Red Hat OpenShift solution architect. He has 15 years of experience in various IT domains, such as public and hybrid multicloud, technical consulting, solution design, implementation, transformation and migration, and data center consolidation for worldwide organizations. He works in IBM Consulting™ as an Infrastructure and Cloud architect, DevOps, and Site Reliability Engineering. He has a BTech degree in Computer Science. His technical skills cover hybrid cloud, Google Cloud Platform, Azure, IBM Cloud, Kubernetes, Red Hat OpenShift, DevOps, security, observability, integration, and open-source software.

**Munshi Hafizul Haque** is a Senior Platform Consultant at Red Hat who is based in Kuala Lumpur, Malaysia. Munshi is an experienced technologist in engineering, design, and the architecture of PaaS and cloud infrastructures. At the time of writing, he is part of the Red Hat Consulting Services team, where he helps organizations adopt automation, container technology, and DevOps practices. He worked for IBM as a senior consultant with IBM Systems Lab Services in Petaling Jaya, Malaysia. He is a specialist in IBM Power and associated enterprise edition technology.

**Subha Hari** is a Senior Delivery Consultant from IBM Technology Expert Labs (Sustainability Software) who is based in Bangalore, India. She has over 19 years of experience, primarily in Performance Testing of the IBM Sterling™® Order Management suite of applications, Production Performance Health Checks, sizing, and high availability and disaster recovery (HADR) activities. She holds a Masters degree (Master of Computer Applications) from Bharathidasan University, Trichy, India. Subha has led various initiatives on automation that used Ansible, Python, and shell scripting. Her areas of expertise include pre-sales, performance testing/benchmarking, and upgrading and modernizing the IBM Sterling suite of products.

**Andrey Klyachkin** is a solution architect at eNFence, an IBM Business Partner who is based in Germany. He has more than 25 years experience in UNIX systems, designing and supporting AIX and Linux systems for different customers worldwide. He is a co-author of many IBM AIX and IBM Power certification courses, and is an IBM Champion and IBM AIX Community Advocate. He is also a Red Hat Certified Engineer and Red Hat Certified Instructor.

**Osman Omer** is a senior IT Managing Consultant who is based in Qatar. He has worked for IBM for 20 years. He worked as a software engineer for 8 years in Rochester, Minnesota before joining Lab Services. He has worked for IBM Systems management, cloud solutions, and automation services. His first project was porting IBM i to be managed by HMC, and then worked on IBM i OS enablement for system management, tools, Systems Director, VMControl, and PowerVC. As a Lab Services consultant, he helps IBM customers with the products that he used to develop. After moving to Qatar, he became a member of the MEA team that is responsible for cloud and automation services delivery in the region. He acts as the EMEA Power Services Delivery Practice Leader in addition to his consulting and leadership responsibilities. Osman holds a master degree in Computer Science from South Dakota State University.

**Rosana Ramos** is a Security Architect at the IBM Systems BISO Organization. She holds a bachelor degree in Computer Engineering from Universidad de Guadalajara México and a master degree in Computer Science from Universidad Autonoma de Guadalajara. She has more than 10 years of experience in Linux and UNIX system administration, with a specialty in implementing security best practices and system hardening. She is certified as a Certified Information Systems Security Professional (CISSP), Certified Ethical Hacker (CEH), Certified in Risk and Information Systems Control (CRISC), and a Master Certified Technical Specialist by the Open Group.

**Prashant Sharma** is an IBM Power Brand Technical Specialist who is based in Singapore. He holds a degree in Information Technology from University of Teesside, England. He has over 12 years of experience in IT Infrastructure Enterprise Solutioning; pre-sales, client, and partner consultation; and technical enablement and implementations relative to IBM Power servers, IBM i, and IBM Storage.

**Stephen Tremain** has been with IBM for 17 years, and works as a Software Engineer at IBM Security® - Australia Development Lab on the Gold Coast in Queensland, Australia. Before joining IBM, Stephen worked as a UNIX System Administrator at an investment bank for 10 years, and also worked in the education and research sectors. Stephen graduated from the University of New England in Australia with a BS and a Graduate Diploma in Agricultural Sciences.

**Perna Upmanyu** is a Software Performance Analyst in the Cognitive Systems Power Servers performance team in India. She holds an M.Tech degree in Software Systems from BITS Pilani. Perna has over 15 years of experience working with customers designing and deploying solutions on IBM Power servers. She focuses on automation and data lakes-based design and deployments. Perna's areas of expertise include system performance, availability, and automation.

**Sundaragopal Venkatraman** (Sundar) is a Red Hat Industry Specialist. He has diversified skills in hybrid cloud automation, application migration, and modernization of Red Hat and IBM portfolios. Sundar has over 23 years of experience working closely with customers to overcome business challenges by using technologies. He has been recognized as a Platinum Author for IBM Redbooks publications. He holds multiple patents, and is an Invention Plateau holder. He has delivered key notes at worldwide conferences on technology transformation and modernization. He is a co-chair for the IT specialist board in the Asia-Pacific region.

Thanks to the following people for their contributions to this project:

Sukanta Basak, Senior Manager - Solution Architecture Partner Ecosystem  
**Red Hat India, Bengaluru**

Jitendra Singh, Specialist Solution Architect Automation and SAP  
**Red Hat India, Bengaluru**

Anant Dhar, Senior Ecosystem Solution Architect  
**Red Hat India, Bengaluru**

Dr Manoj Kumar Jain, Automation Specialist Solution Architect  
**IBM India**

Kanan Ganjoo, Customer Success Manager Architect - Observability  
**IBM India, Bangalore**

Bhargavaram Akula, WW ISV Engineering  
**IBM India, Hyderabad**

Shiva Laveti, Systems Engineer - ISV Engineering Infrastructure  
**IBM India, Bangalore,**

Benoit Marolleau, Senior Solution Architect - IBM Client Engineering EMEA  
**IBM France, Montpellier**

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:  
<https://www.linkedin.com/groups/2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:  
<https://www.redbooks.ibm.com/subscribe>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:  
<https://www.redbooks.ibm.com/rss.html>





# Introducing Ansible and IBM Power

This chapter describes the need for automation in complex IT environments and some of the technologies and tools that are available to bring the benefits of automation to your business. We describe Ansible, and why it is considered the most versatile automation solution.

We also describe IBM Power, which is the IBM powerful and robust midrange server platform. We provide an overview of IBM Power and show how your IBM Power servers can be automated by using the same Ansible tools that you might be using for other servers, storage devices, and networking components in your environment.

We also explore how Ansible automation can reshape the IT management landscape by providing an end-to-end automation platform to configure systems, deploy software, and orchestrate workflows on IBM Power. We delve into provisioning, patch management, security, configuration, business continuity, disaster recovery (DR), and application to showcase how Ansible and IBM Power harness the combined potential of cutting-edge technology and a highly configurable automation platform.

The following topics are described in this chapter:

- ▶ Why automation
- ▶ Automation tools and techniques
- ▶ Understanding Ansible: A powerful automation tool
- ▶ Introducing IBM Power
- ▶ Ansible for Power

## 1.1 Why automation

The dictionary defines automation as “the technique of making an apparatus, a process, or a system operate automatically.” We define automation as “the creation and application of technology or programs to replace repeatable tasks with minimal human intervention.”

Automation in everyday life has been a staple for many years, for example:

- ▶ Automatic dishwashers do our dishes and automatic washers and dryers clean our clothes.
- ▶ Robotic machines do repetitive tasks in manufacturing.
- ▶ Machines automatically “call home” when an error is detected.
- ▶ Features in your automobile automatically check and report on safety issues and can even enable the car to drive itself.

Automation is designed to do simple, repetitive tasks that consume time and energy in everyday life. However, automation has become a necessity in information technology because the number of components that must be managed is increasing exponentially.

Here are some benefits of automation:

- ▶ Perform processes that are difficult to be done manually.  
IT automation tools help you perform different tasks that are difficult to perform manually. For example, provisioning and deploying environments manually is a laborious process that requires highly skilled personnel with hands-on knowledge. Using automation to apply Infrastructure as Code (IaC) enables your IT team to provide self-service capabilities to developers. It delivers preapproved resources and configurations quickly, on demand, and without manual intervention.
- ▶ Create solutions that work consistently across different technologies or cloud platforms.  
IT team tasks are complex in hybrid and multicloud environments. Every cloud provider has their own tools and methods, which rarely work well with each other and makes life difficult for IT teams because they must manage each cloud separately and in a different manner. You can create automation assets by codifying resources across all clouds. You can have a single application programming interface (API) for an operation regardless of the cloud platform, which helps IT teams operate more efficiently and effectively.
- ▶ Keep pace with increasing infrastructure needs.  
Infrastructure needs grow rapidly, so IT teams strive hard to manage new requirements, especially if the staffing levels are low. IT automation tools help the IT team handle this situation by eliminating or streamlining a vast array of manual tasks and processes.
- ▶ Integrate and deploy applications quickly with zero downtime.  
For modernization, fast and reliable application development is crucial. The continuous integration and continuous deployment (CI/CD) approach helps organizations deliver applications more swiftly and with minimal errors. By adopting a CI/CD pipeline that applies automation throughout the application lifecycle, including integration, testing, delivery, and deployment, the teams can produce stable applications more quickly and with zero downtime.
- ▶ Produce trusted, secured, and compliant applications.  
When there is an automated CI/CD pipeline with security gates, you can be assured that trusted software is produced for deployment. This automation helps streamline daily operations and integrate security and compliance into the processes from the beginning.

- ▶ Construct remediation processes.

When there is a security breach, they should be detected and contained as quickly as possible. If there are multiple systems and platforms, then it is a complicated task to apply fixes manually, and they can be error-prone too. Every second matters during a security breach, and automation helps a security team apply remediation to affected systems across all environments more quickly with fewer chances for errors.

- ▶ Get more time to focus on competitive and novel initiatives.

Most IT tasks can be automated to some extent. Automating even a few of them can help reduce the amount of time that IT teams spend on manual processes. The more automation that is used, the more time the teams must work on futuristic and innovative projects.

- ▶ Eliminate human errors.

Manual routine tasks can be error-prone. With automation, they can be eliminated and you can expect predictable and consistent results.

- ▶ Get to know the operational complexities and costs.

Although automation helps to streamline and manage complex tasks, it can also provide operational analytics, which can help you understand and reduce the costs that are involved.

- ▶ Transforms organizations.

When automation becomes second nature to an organization, the teams can save time and money, and have more time to work on strategic initiatives. It increases productivity and decreases costs due to human errors. It increases employee satisfaction because manual and repetitive tasks are boring and laborious. Happier employees, less error-prone processes, and cost and time savings makes a successful organization.

### 1.1.1 Orchestration versus automation

*Automation* generally applies to doing a single process, task, or a few related tasks. For example, checking whether some systems need updates or whether updates were performed.

*Orchestration* refers to managing multiple automated tasks to create a dynamic workflow. For example, checking whether if your systems need updates; if the update is needed, perform the update on each system; and if a restart is needed, perform the restart or inform the user that a restart is needed or is scheduled for later.

The two concepts are closely related, and at times the line between the two concepts can be blurry. The biggest differentiator is that orchestration takes a set of automated tasks and groups them; checks for values before and after a task completes; checks the results for each task (was it successful or not); adds intelligence to the workflow; and adapts the steps based on results from each step.

You can consider automation as a subset of orchestration where you cannot orchestrate manual, non-automated tasks, but multiple automation tasks strung together are not orchestration unless they include programmatic control over the process based on the results of each task.

Automation and orchestration are not meant to replace the role of the system administrator, but aim to help them create more reliable automated tasks and provide time to focus on innovation, problem solving, or studying new technologies instead of day-to-day manual tasks.

Figure 1-1 illustrates the differences between automation and orchestration.

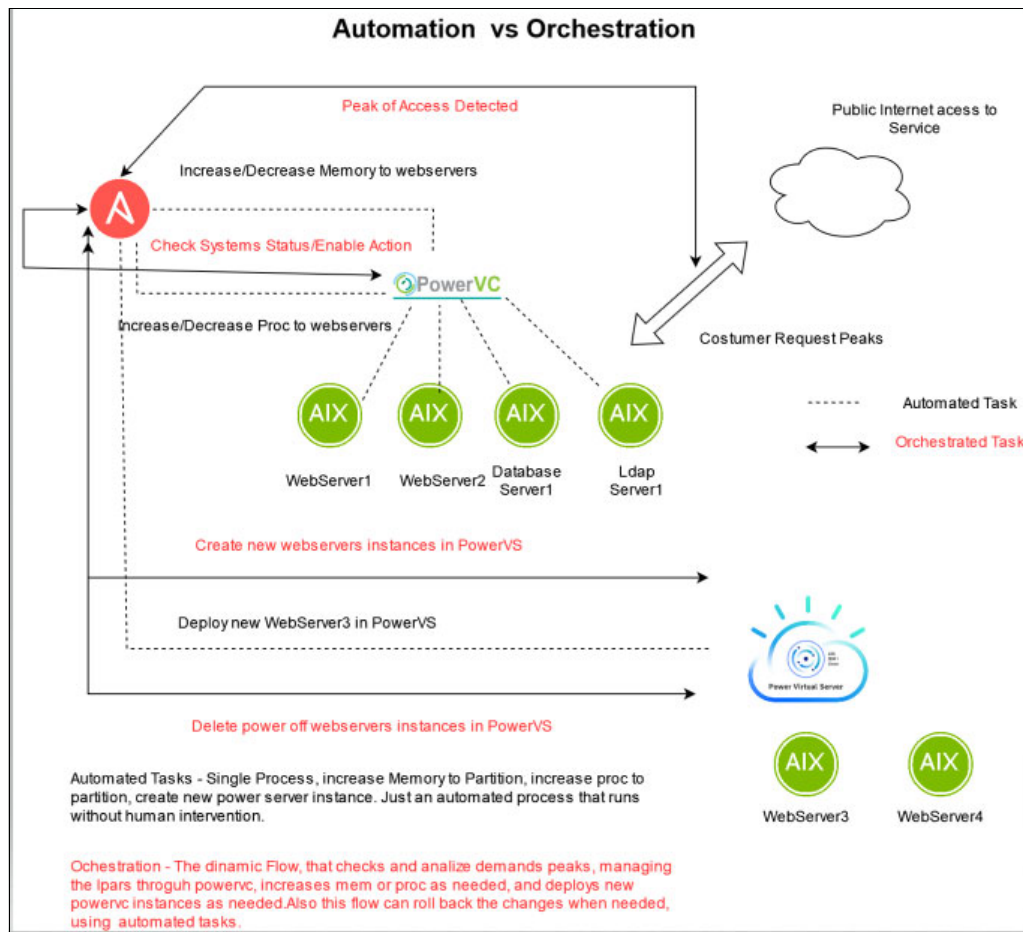


Figure 1-1 Automation versus orchestration

## 1.2 Automation tools and techniques

Automation of IT processes has been a focus for many years now, and many tools and techniques have been developed that companies are using. Then, the COVID-19 pandemic arrived in 2020, which accelerated the pace of automation because many companies were forced to start using automation to address the problems that were caused by the crisis.

## 1.2.1 Common IT automation tools

There are many tools that are available for automation in the IT environment. Here is a list of some of the more widely used tools:

- ▶ Chef (now Progress Chef) is a configuration management tool that is written in Ruby and Erlang. It uses a pure-Ruby, domain-specific language (DSL) for writing system configuration “recipes”. Chef is used to streamline the task of configuring and maintaining a company's servers and can integrate with cloud-based platforms such as Amazon EC2, Google Cloud Platform, Oracle Cloud, OpenStack, IBM Cloud, Microsoft Azure, and Rackspace to automatically provision and configure new machines.
- ▶ Puppet is an automated administrative engine for your Linux, UNIX, and Windows systems that performs administrative tasks (such as adding users, installing packages, and updating server configurations) based on a centralized specification. You can manage and automate infrastructure and complex workflows with reusable blocks of self-healing IaC, and quickly deploy infrastructure to support your evolving business needs at will (and at scale) with model-driven and task-based configuration management.
- ▶ HashiCorp Terraform is an IaC tool that you can use to define both cloud and on-premises resources in human-readable configuration files that you can version, reuse, and share. You can use a consistent workflow to provision and manage your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, and high-level components like DNS entries and Software as a Service (SaaS) features.

In August of 2023, HashiCorp announced that future versions of Terraform will be covered by the business source license (BSL) compared to current versions being open-source under the Mozilla Public License (MPL) 2.0 license. Therefore, the Linux Foundation announced the formation of OpenTofu, an open source alternative to Terraform for code provisioning.

- ▶ Ansible is an open-source, cross-platform tool for resource provisioning automation that DevOps professionals use for continuous deployment (CD) of software code by leveraging an “IaC” approach. The Ansible automation platform has evolved to deliver sophisticated automation solutions for operators, administrators, and IT decision-makers across various technical disciplines. It is an enterprise automation solution with flourishing open-source software. It operates on several UNIX like platforms, and can manage systems like UNIX and Microsoft architectures. It comes with descriptive language for describing system settings.

Because of the broad acceptance of the Ansible platform, its open-source design, and its wide support for many devices and platforms, it is becoming a dominant tool in the market. However, it is also common to use other automation tools with Ansible to do more complex automation. For example, many companies use Ansible with Terraform to provide automatic provisioning of their infrastructure.

## 1.3 Understanding Ansible: A powerful automation tool

Ansible is a versatile, open-source automation platform that can help streamline IT operations. It offers a comprehensive set of features for configuring systems, deploying software, and orchestrating complex workflows. Ansible core strengths are its simplicity and focus on security and reliability.

## 1.3.1 Ansible architecture

As shown in Figure 1-2, the Ansible architecture consists of an Ansible Controller and one or more Ansible client hosts. The controller runs automation tasks and houses Ansible collections, which contain modules, plug-ins, and roles defining the actions Ansible can perform on client nodes.

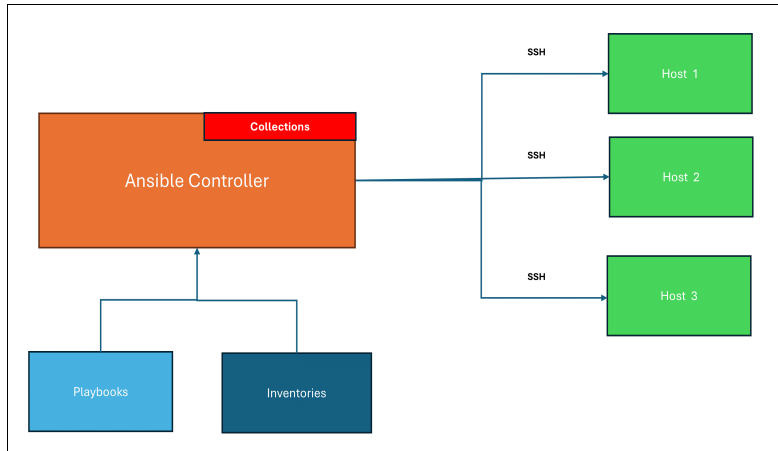


Figure 1-2 Simplified Ansible architecture

### Playbooks: The heart of Ansible Automation

Ansible playbooks are YAML files that define sequences of tasks to run on remote hosts. These tasks can range from installing packages to configuring services or copying files. Playbooks enable IT teams to automate infrastructure provisioning, configuration management, application deployment, and more.

### Why choose Ansible

Ansible offers numerous benefits for IT professionals seeking to improve efficiency, scalability, and consistency in their infrastructure. Here are some key advantages:

- ▶ **Versatility:** Ansible supports a wide range of devices and can scale to accommodate growing environments and automation needs.
- ▶ **Agentless architecture:** Ansible manages devices by using Secure Shell (SSH), which eliminates the need for agents on target systems.
- ▶ **Flexibility:** Ansible can be used for simple CLI tasks and complex workflows that are defined in playbooks.
- ▶ **Extensive module library:** Ansible provides a rich collection of modules for managing various systems, cloud infrastructures, and OpenStack.
- ▶ **Declarative approach:** With the Ansible declarative syntax, you can define the state of a system, and Ansible takes the necessary steps to achieve it.
- ▶ **Ease of learning:** The Ansible YAML syntax and minimal learning curve make it accessible to IT professionals at all levels.

Ansible is a powerful automation tool that can help organizations improve efficiency, scalability, and reliability in their IT infrastructure. By leveraging Ansible playbooks, IT teams can streamline routine tasks, automate complex workflows, and help ensure consistent configurations across their environments.

## 1.3.2 Options for implementing Ansible

As you decide to implement Ansible for IT management, it is essential to select the correct product and support level to meet your organization's needs. This section describes some of the options that are available to you.

### Ansible Community

The community versions of Ansible primarily include the following ones:

- ▶ Ansible Core

Ansible Core is a fundamental part of Ansible. It provides the core automation engine. It is an open-source tool that includes the basic functions for configuration management, application deployment, and task automation. Ansible Core includes modules, plug-ins, and the CLI that is needed to run playbooks and manage configurations.

- ▶ AWX

AWX is the upstream, open-source project that serves as the community version of Red Hat Ansible Tower. AWX provides a web-based UI, Representational State Transfer (REST) API, and task engine for managing Ansible automation at scale. AWX offers role-based access control (RBAC), job scheduling, graphical inventory management, and more. It helps users manage and scale automation efforts.

**Note:** Although AWX is available at no charge, it does not come with enterprise-level support or guarantees.

- ▶ Ansible Collections

Ansible Collections are pre-packaged modules, roles, and plug-ins that are created and shared by the community. With Collections, users can extend Ansible functions with more content that is often maintained by the community or specific organizations. Collections can be downloaded from Ansible Galaxy, a community hub for sharing and discovering Ansible content.

- ▶ Ansible Galaxy

Ansible Galaxy is a repository for sharing and discovering Ansible roles and collections. It is a community-driven platform where users can find reusable Ansible content to simplify automation tasks. It provides a searchable repository of roles and collections that are created by the Ansible community, which can be integrated into your automation workflows.

These community versions are suitable for individual users, small teams, and development environments but lack the formal support and advanced features that are provided by Red Hat Ansible Automation Platform.

### Ansible Automation Platform

Ansible Automation Platform is a subscription-based enterprise solution that combines over 20 community projects into a fully supported automation platform. Ansible Automation Platform provides curated, certified, and validated Ansible Collections and roles from partners like IBM, Juniper, Cisco, and public cloud providers.

Here are the key considerations for choosing Ansible Automation Platform:

- ▶ **Support level:** Ansible Automation Platform offers enterprise-grade support, which includes SLAs for security, compatibility, and upgrades. Community options might have limited support.
- ▶ **Features:** Ansible Automation Platform includes features beyond Ansible Core, such as a web interface and integration with other tools.
- ▶ **Cost:** Ansible Automation Platform is a subscription-based product, but community options are available at no charge.
- ▶ **Scale and complexity:** For large organizations with complex automation needs, Ansible Automation Platform might be the better choice due to its enterprise-grade features and support.

By carefully evaluating these factors, you can select the Ansible offering that best aligns with your organization's goals, budget, and support requirements.

Table 1-1 compares these offerings.

*Table 1-1 Comparing Ansible offerings*

<b>Technology</b>	<b>Community/Upstream</b>	<b>Supported/Downstream</b>
Ansible Core	X	
Community Ansible	X	
AWX	X	
Ansible Automation Platform		X

Which method you use to procure Ansible is determined by your business requirements. If your automation environment is small and not business critical, it is acceptable to use the community-supported versions. However, if you are supporting business-critical environments, consider the benefits of a supported enterprise product. Consider an enterprise solution if you have the following requirements:

- ▶ Require enhanced security.
- ▶ Are embarking on an IT transformation initiative.
- ▶ Are ready to expand automation to include more people, teams, and use cases.
- ▶ Need flexibility to adapt to changing business requirements-with proven, innovative solutions.
- ▶ Want to prioritize automation objectives over managing automation infrastructure.

***Which Ansible option is right for my organization***

Community Ansible is suitable for individuals and small teams seeking automation for personal workloads or home lab environments. For collaborative automation efforts, AWX or Ansible Automation Platform offer more robust options.

Although AWX is a no-charge, open-source project, it lacks enterprise-grade support, such as SLAs for security, compatibility, and upgrade migrations. This lack of support can lead to hidden costs that are associated with security breaches and time-consuming fixes. However, AWX can be valuable for small-scale labs, developers contributing to the upstream code, or as a sandbox for learning Ansible Automation Platform before migrating to an enterprise solution.



For organizations aiming to scale automation at an enterprise level, Ansible Automation Platform is a more comprehensive choice. It offers developer tools, flexible deployment options, and SLAs for compatibility, upgrades, and security. Ansible Automation Platform also provides transparent and efficient scaling of automation investments.

Table 1-2 outlines the key capabilities of each option to help you determine whether Community Ansible, AWX, or Ansible Automation Platform best aligns with your organization's needs.

*Table 1-2 Community Ansible and AWX compared to Ansible Automation Platform*

<b>Capability</b>	<b>Community Ansible and AWX</b>	<b>Ansible Automation Platform</b>
Security	Not available	Trusted chain-of-custody for certified and private content.
Certified content and partner ecosystem	Not available	140+ certified content collections across 60+ partners. Benefit from prebuilt, fully supported, and certified automation content from Red Hat and partners.
Lifecycle support	Not available	At least 18 months of enterprise support per release. Critical bugpatch and security vulnerability back porting for all components.
Legal protections	No protections	Intellectual property protections through the Open Source Assurance Agreement.
Analytics	Not available	Automation analytics and Red Hat Insights for Ansible Automation Platform offer in-depth analytics and reporting for planning and tracking performance and adoption.
Upgrades and migrations	Not supported	Supported migration to major releases and upgrades to minor releases.
Training and consulting	Not available	Expert resources to help you build and run a successful automation practice that is backed by robust training offerings and support. Hands-on migration assistance from AWX to the Ansible Automation Platform is also available.
Cloud deployment options	Not available	Managed and self-managed applications are available to deploy on your cloud of choice, which includes Microsoft Azure, AWS, and Google Cloud. Counts toward committed expense agreements. Supported by Red Hat with integrated billing. View deployment options and pricing information.
Event-Driven Ansible	Separate upstream project that requires manual integration into your environment.	Event-Driven Ansible is an integrated and tested product component of the Ansible Automation Platform that reduces manual tasks, delivers more efficient IT operations, and frees your teams to focus on innovation.
Private Automation Hub	Separate upstream project that requires manual integration into your environment.	Private Automation Hub is an integrated and tested product component of Ansible Automation Platform.
IBM watsonx Code Assistant for Red Hat Ansible Lightspeed	A generative artificial intelligence (AI) service for task creation that is available to all Ansible users.	A full commercial version of Ansible Lightspeed. IBM watsonx Code Assistant for Red Hat Ansible Lightspeed is available for Ansible Automation Platform subscribers.

When considering automation to address resource constraints, organizations must evaluate their readiness to manage disparate tools and their interest in contributing to open-source development models.

Ansible Automation Platform offers a distinct advantage with Event-Driven Ansible. This feature automates tasks by connecting events to corresponding actions through rules. By defining rulebooks, you enable Event-Driven Ansible to automatically recognize events, match them to appropriate actions, and run them. This approach frees your teams to focus on high-value work.

Ansible Automation Platform is a trusted enterprise solution with a proven track record. It is used by over 3,000 global customers across various industries to create, manage, and scale IT automation. As a comprehensive, integrated solution, it combines open-source innovation with enterprise-hardened features to boost productivity and accelerate project completion.

### 1.3.3 Ansible Automation Platform

Ansible Automation Platform is a robust, end-to-end automation platform that helps you streamline IT operations. Built on open-source innovation and hardened for enterprise environments, Ansible Automation Platform offers a comprehensive set of tools for configuring systems, deploying software, and orchestrating advanced workflows. It empowers organizations to create, manage, and scale automation across their entire enterprise.

#### **End-to-end automation**

Ansible Automation Platform is a strategic automation solution that is used by every Fortune 500 company in the airline, government, and military sectors to create, manage, and scale automation strategies.<sup>1</sup>

#### ***Creating and sharing***

Developers can write consistent code in an execution environment without constraints or operational overhead by using the following features:

- ▶ Event-Driven Ansible can automate IT actions with user-defined, rule-based constructs and create end-to-end automated scenarios for use cases across the IT landscape.
- ▶ Ansible content collections contain Red Hat Ansible Certified Content and Ansible validated content that is reusable code to automate faster.
- ▶ Ansible content tools help developers create consistent code, streamlining the building and deployment of execution environments to speed development cycles and realize value quicker.
- ▶ Automation execution environments are container images that are used to run Ansible Playbooks and roles.

#### ***Managing and measuring***

Operations teams can automate from development through production by using the following tools:

- ▶ Ansible Automation Hubs, where you can find, download, and share supported collections. Private Automation Hub is a curated library of automation content for internal teams to share.
- ▶ An automation controller is a centralized management tool to manage inventory, launch and schedule workflows, track changes, and integrate reporting with a centralized UI.

---

<sup>1</sup> Source: <https://www.redhat.com/en/topics/automation/why-choose-red-hat-for-automation>

- ▶ The Ansible Automation Platform trusted supply chain offers enhanced security and compliance. For more information, see [What's new in Ansible Automation Platform 2.3](#).
- ▶ Automation analytics and Red Hat Insights for Ansible Automation Platform provide rich reporting and advanced analytics to optimize your automation, proactively identify potential issues, mitigate vulnerabilities, and improve resolution times.

### ***Scaling up and out***

As more people start to use automation tools, IT architects can design and expand automation tactics across multiple environments and geographies by using an automation mesh, which can scale automation of large inventories across diverse network topologies, platforms, and regions. Flexible design options help you deliver automation across physical and virtual data centers, hybrid cloud environments, and edge locations, localizing automation execution to improve network performance. An automation mesh helps your organization operate at a global scale.

Ansible Automation Platform is a single automation platform for multiple use cases:

- ▶ **Hybrid cloud:** Automate cloud-native environments and manage infrastructure and services across public, private, and hybrid clouds with certified integrations.
- ▶ **Edge:** Standardize configuration and deployment across your entire IT landscape (from data center to cloud to edge environments) with a single, consistent automation platform.
- ▶ **Networks:** Manage entire network and IT processes across physical networks, software-defined networks, and cloud-based networks to edge locations.
- ▶ **Security:** Orchestrate security systems by using a curated collection of modules, roles, [research from IDC](#), and playbooks to investigate and respond to threats.
- ▶ **Infrastructure:** Consistently deploy, manage, and scale infrastructure workloads where it makes the most sense in your physical data center, private or public cloud, or at the network edge.
- ▶ **Provisioning:** Streamline the process of PXE starting and kickstarting bare-metal servers or VMs, and creating virtual or cloud instances from templates.
- ▶ **Configuration management:** Centralize configuration file management and deployment with a low learning curve for administrators, developers, and IT managers.

For more information, see [Red Hat Ansible Automation Platform: A beginner's guide](#).

## Ansible Automation Platform core components and architecture

Figure 1-3 shows the Ansible Automation Platform components and architecture.

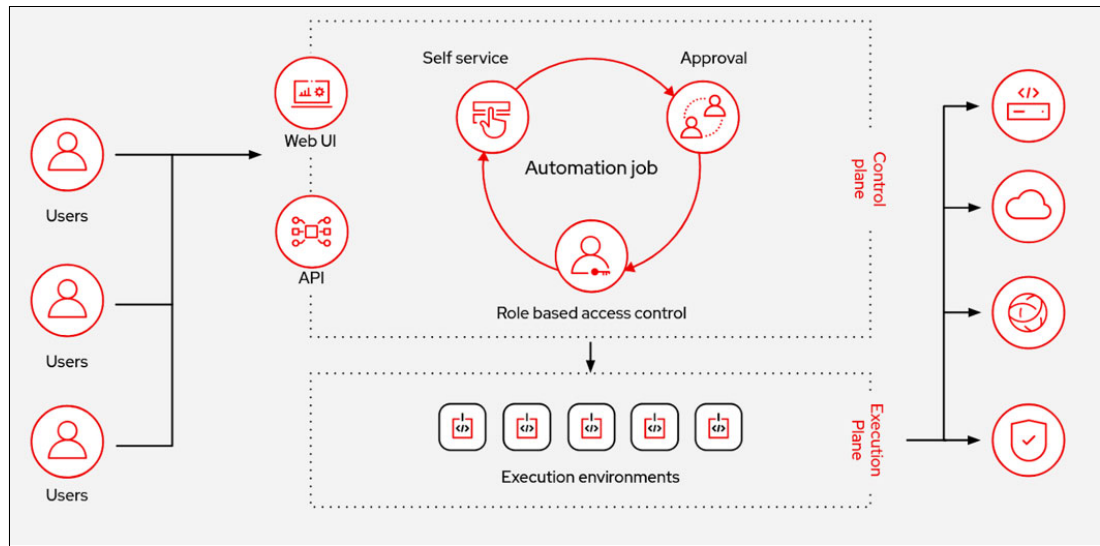


Figure 1-3 Ansible Automation Platform core components<sup>2</sup>

### Automation controller

The automation controller is a distributed system where different software components can be colocated or deployed across multiple compute nodes. Automation controller is the control plane for automation, and includes a UI, browse-able API, RBAC, job scheduling, integrated notifications, graphical inventory management, CI/CD integrations, and workflow visualizer functions. Manage inventory, launch and schedule workflows, track changes, and integrate into reporting, all from a centralized UI and RESTful API.

In the installer, node types of control, hybrid, execution, and hop are provided as abstractions to help the user design the topology that is appropriate for their use case.

### Automation hub

With the automation hub, you discover and use new certified automation content from Red Hat Ansible and Certified Business Partners. On the Ansible Automation Hub, you can discover and manage Ansible Collections, which are supported automation content that is developed by Red Hat and its partners for use cases such as cloud automation, network automation, and security automation.

Private Automation Hub provides both disconnected and on-premises solutions for synchronizing collections and execution environment images from a Red Hat cloud automation hub. You can also use other sources such as Ansible Galaxy or other container registries to provide content to your Private Automation Hub. Private automation hubs can integrate into your enterprise directory and your CI/CD pipelines.

Figure 1-4 on page 13 shows the development cycle for an automated execution environment.

<sup>2</sup> Source: <https://www.ansible.com/blog/peeling-back-the-layers-and-understanding-automation-mesh>

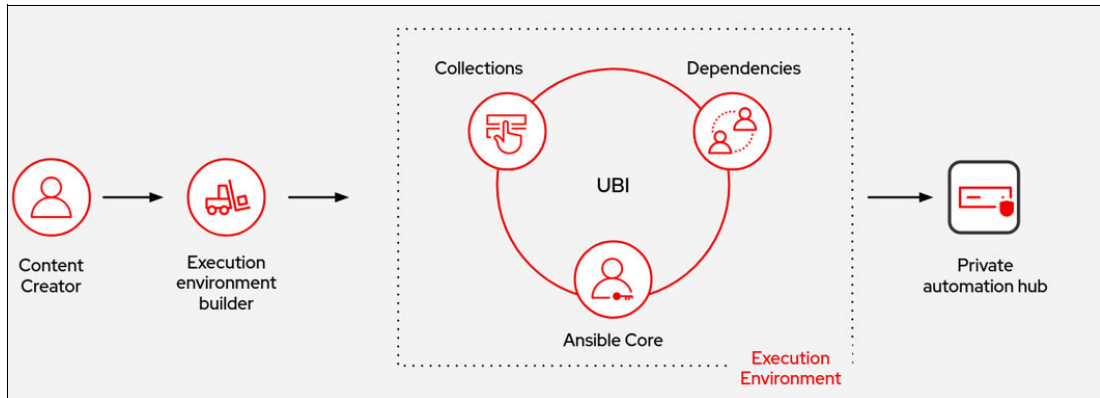


Figure 1-4 Development cycle of an automation execution environment<sup>3</sup>

### Execution environment

An automation execution environment is a container image that is used to run Ansible playbooks and roles. Automation execution environments provide a defined, consistent, and portable way to build and distribute your automation environment between development and production. With execution environments, Ansible Automation Platform administrators can provide and manage the automation environments that meet the needs of different teams, such as networking and cloud teams. These environments also enable automation teams to define, build, and update their automation environments themselves. Execution environments provide a common language to communicate automation dependency between automation developers, architects, and platform administrators.

Figure 1-5 details the automation execution environment.

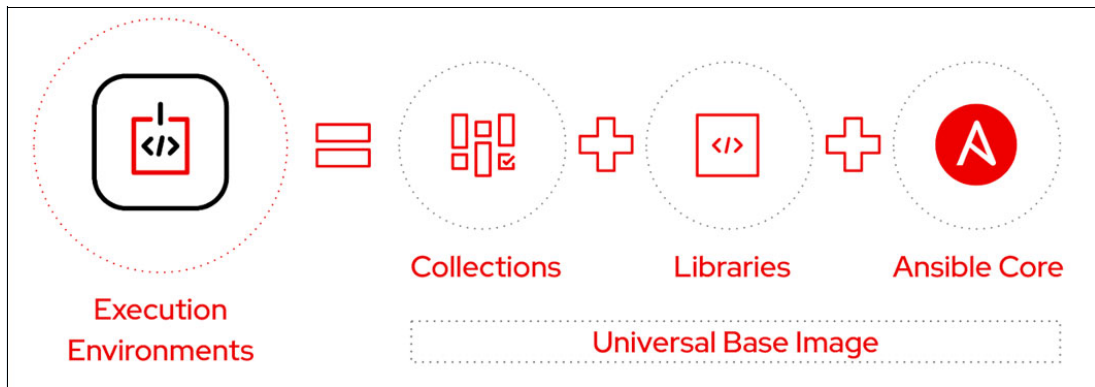


Figure 1-5 Anatomy of automation execution environment<sup>4</sup>

### Ansible Core (Ansible Engine)

Ansible Engine is an implementation in which the strategy has slightly changed. Instead of shipping a “kitchen sink” package that is repackaged from the upstream Ansible Project, going forward, Ansible Automation Platform ships the `ansible-core` package as a stand-alone Red Hat Package Manager and within execution environments.

<sup>3</sup> Source: <https://cloudnroll.com/2023/05/22/understanding-and-creating-ansible-execution-environments/>

<sup>4</sup> Source: <https://www.openvirtualization.pro/whats-new-in-red-hat-ansible-automation-platform-2/>

### Automation mesh

An automation mesh is an overlay network that is intended to ease the distribution of work across a large and dispersed collection of workers through nodes that establish peer-to-peer connections with each other by using existing networks. An automation mesh uses unique node types to create both the control and execution plane, as shown in Figure 1-6.

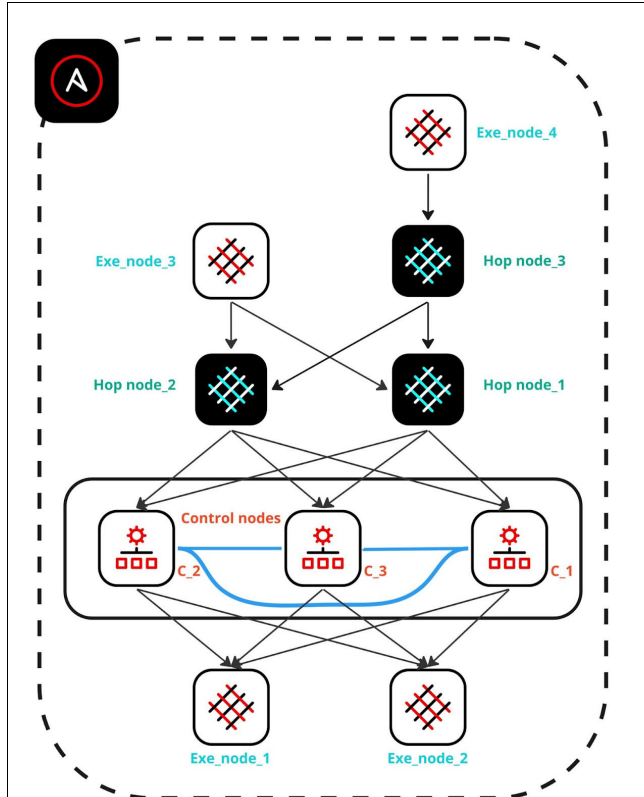


Figure 1-6 Sample automation mesh networking

### Control plane

The control plane consists of hybrid and control nodes:

- ▶ Hybrid nodes: The default node type for control plane nodes. They are responsible for automation controller runtime functions like project updates, management jobs, and `ansible-runner` task operations. Hybrid nodes are also used for automation execution.
- ▶ Control nodes: The control nodes run project and inventory updates and system jobs, but not regular jobs. Execution capabilities are disabled on these nodes.

### Execution plane

The execution plane consists of execution nodes that run automation on behalf of the control plane and have no control functions, and consist of hop and execution nodes.

- ▶ Execution nodes: Run jobs under `ansible-runner` with Podman isolation. This node type is similar to isolated nodes. It is the default node type for execution plane nodes.
- ▶ Hop nodes: Similar to a jump host, hop nodes route traffic to other execution nodes. Hop nodes cannot run automation.
- ▶ Peer relationship: Defines the node-to-node connections between controller and execution nodes.

For more information, see [Ansible Automation Platform](#).

### 1.3.4 Event-driven automation

Event-driven automation (EDA) is the process of responding automatically to changing conditions in an IT environment to help resolve issues faster and reduce routine and repetitive tasks.<sup>5</sup>

EDA helps connect data, analytics, and service requests to automated actions so that activities, such as responding to an outage or adjusting some aspect of an IT system, can take place in a single, rapid motion. Automating in an “if-this-then-that” fashion helps IT teams manage how and when to target specific actions.

Figure 1-7 shows a typical EDA environment.

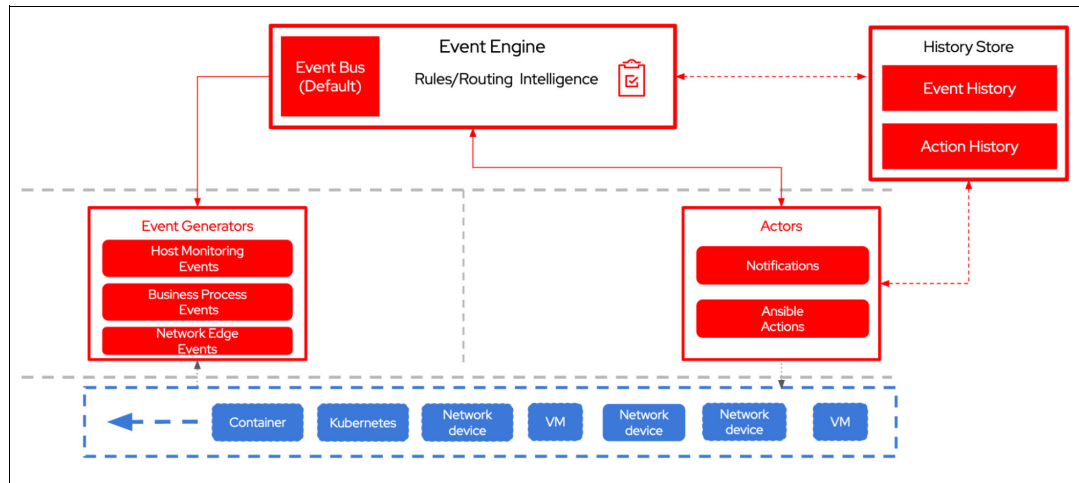


Figure 1-7 Typical event-driven environment<sup>6</sup>

#### What is an IT event

An *event* refers to any detectable occurrence that has significance for the management of an IT infrastructure or the delivery of an IT service. Events are often identified by third-party monitoring tools, and typically indicate significant occurrences or changes of state in applications, hardware, software, cloud instances, or other technologies.

#### What does it mean to be event-driven

In an IT environment, being event-driven means connecting data and service requests to automated actions so that manual steps that are typically taken by IT teams can happen in a single automated workflow. With EDA, systems can initiate a predefined automated response when an event occurs.

For example, a system outage can trigger an event that automatically runs a specific action, such as logging a trouble ticket, gathering facts that are needed for troubleshooting, or performing a restart. Because these actions are predefined and automated, they can be performed more quickly than if the required steps were done manually.

<sup>5</sup> Source: <https://www.redhat.com/en/topics/automation/what-is-event-driven-automation>

<sup>6</sup> Source: <https://www.redhat.com/en/blog/achieving-speed-and-accuracy-through-event-driven-automation>

### ***What does event-driven automation offer IT teams***

As organizations strive to use automation more strategically across hybrid cloud environments and edge locations, they often start by automating IT actions that are central to management and service delivery. Although automation can increase the speed and agility of these processes and minimize human error, some events still require manual troubleshooting and information gathering, which can delay resolution and disrupt everyday operations.

EDA can help teams move from a reactive to a proactive approach to IT management and streamline IT actions with full end-to-end automation. Solutions with event-handling capabilities extend the usage of automation across domains, processes, and geographies, which advances automation maturity by helping ensure operational consistency, resilience, and efficiency.

EDA can help IT teams to achieve the following goals:

- ▶ Select ideal tasks to automate, and then IT domain experts, such as a network engineer, flexibly apply automation to key needs.
- ▶ Build existing operational knowledge into automated decision-making and actions.
- ▶ Complete repetitive tasks efficiently and deliver services more quickly.
- ▶ Reduce low-level tasks and use valuable resources for other priorities.
- ▶ Address problems rapidly before they become urgent issues.
- ▶ Automate repetitive tasks for networking, edge, infrastructure, DevOps, security, and cloud.

### ***Event-driven automation use cases***

Getting started with EDA begins with identifying repetitive, mundane tasks that IT teams complete manually and frequently. Some common use cases include the following ones:

- ▶ **Automated remediation:** EDA can connect the analytics or tickets that flag an issue to the automated steps that will resolve it. Teams can automate the resolution of tickets, the remediation of issues that are based on known system-behavior patterns, or the response to monitored events, such as an alert that a system needs more capacity.
- ▶ **Ticket enrichment:** EDA can contact relevant systems, gather data, and update corresponding tickets with the rich detail that is needed for a thorough effective root cause analysis (RCA) process.
- ▶ **Automated platform scaling:** Application workloads and platforms rely on automated provisioning to help ensure business continuity and reduce the potential impact on customers. Rather than waiting for manual provisioning, IT teams can combine capacity and performance metrics with EDA to automatically provision containers, cloud infrastructure, virtual machines (VMs), and other technologies.
- ▶ **Risk mitigation:** With EDA, security responses can be started when a risk is identified. For example, if a risk is identified on a firewall, an event-driven solution can immediately close down the firewall and create a service ticket, which reduces the opportunity for exposure to a security breach.



- ▶ **Automated tuning and capacity management:** Ongoing tuning and capacity management are necessary for many IT functions, such as managing web applications and monitoring storage pools. For some teams, tuning occurs thousands or tens of thousands of times per month, making it time-consuming when done manually. EDA can respond to these types of events based on predetermined rules to address things like low storage capacity and trigger automatic adjustments.
- ▶ **Scaling automation:** As with tuning, it can be burdensome to manually scale applications' storage, processing, and network bandwidth to meet user demand. For example, an EDA solution can monitor buffer pools, automatically adjusting sizes as limits are reached.

For more information, see [What is event-driven automation](#).

### 1.3.5 Infrastructure as Code: integration of Ansible and Terraform

In the modern landscape of cloud computing and DevOps practices, IaC has emerged as a pivotal concept. Two of the most prominent tools in this realm are Ansible and Terraform. Although both tools contribute to automating infrastructure management, they do so with distinct philosophies and purposes. This section describes the differences and overlaps in features between Ansible and Terraform and sheds light on when and how to best leverage each tool.

#### **Ansible**

Ansible is known for its focus on orchestration and configuration management. It excels at automating tasks that involve the setup, configuration, and management of systems, applications, and networks. Ansible uses a declarative approach, where you define the state of your systems, and Ansible brings them to that state. Ansible playbooks, which are written in YAML, encapsulate these declarative configurations and automation workflows.

#### ***Configuration management***

Ansible shines in helping ensure consistency across various systems. Its idempotent nature helps ensure that tasks run only if necessary, which reduces the risk of unintended changes.

#### ***Ad hoc commands***

With Ansible, you can perform quick and flexible running of ad hoc commands across multiple servers. This feature is useful for tasks that require immediate attention or investigation.

#### ***Overlap with Terraform***

Although Ansible can manage provisioning tasks, it is not designed as a full-fledged infrastructure provisioning tool like Terraform. However, Ansible and Terraform can complement each other by enabling Ansible to handle complex configuration tasks after Terraform provisions the infrastructure.

#### **Terraform**

Terraform is designed for provisioning and managing infrastructures. It employs a declarative language to define the infrastructure's state, which creates a clear separation between the “what” and the “how.” Terraform configuration files, which are written in HashiCorp Configuration Language (HCL), describe the infrastructure resources, their dependencies, and relationships.

#### ***Infrastructure provisioning***

The strength of Terraform is its ability to create and manage infrastructure resources across various cloud providers and on-premises environments. It excels at managing the lifecycle of resources, from creation to updates and destruction.

### ***State management***

Terraform maintains a state file that records the state of the infrastructure. This state file enables Terraform to determine what changes are necessary to reach another state and helps prevent accidental changes.

### ***Overlap with Ansible***

Although Terraform can provision infrastructure, it is not designed for handling configuration management tasks like Ansible. However, Terraform and Ansible can work together by leveraging Ansible capabilities to configure the provisioned infrastructure.

### **Complementary roles**

Ansible and Terraform are not mutually exclusive; in fact, they often work best when used in tandem. Terraform excels at setting up the infrastructure, which helps ensure that resources are created and managed accurately, and Ansible configures and maintains the systems running on that infrastructure. This synergistic approach maximizes the strengths of both tools while minimizing their respective weaknesses. Red Hat has created two certified collections to help you to better integrate the two tools:

- ▶ The [Terraform Collection for Ansible Automation Platform](#) automates the management and provisioning of IaC by using the Terraform CLI tool within Ansible playbooks and Execution Environment run times.
- ▶ With the [Ansible provider for Terraform](#), you can integrate your Terraform workflows into your Ansible workflows by collecting the build results to populate an Ansible inventory for further automation with Ansible.

Ansible and Terraform offer distinct yet complementary features in the world of IaC. Ansible excels at orchestration and configuration management, and Terraform focuses on provisioning and managing infrastructure. By understanding their differences and overlaps, DevOps practitioners can harness the power of both tools to create a robust, automated, and efficient infrastructure management.

### **IBM Cloud provisioning**

Running your infrastructure in the cloud also benefits from IaC because you can automate provisioning and management. IBM Cloud provides a service for infrastructure provisioning of environments in the IBM Cloud by using both Terraform and Ansible.

### ***IBM Cloud Schematics***

IBM Cloud Schematics is a managed service that offers both Terraform and Ansible in a unique way where you can provision infrastructure by using Terraform and trigger Ansible configuration management in the same deployment. Most IBM Cloud resources across IBM Cloud Classic, IBM Virtual Private Cloud, and IBM Power Systems Virtual Server can be provisioned by using Terraform and Ansible.

## 1.3.6 Provisioning

Automated infrastructure provisioning is the first step in automating the operational lifecycle of your applications. From traditional servers to the latest serverless or function-as-a-service environments, Ansible Automation Platform can provision cloud platforms, virtualized hosts and hypervisors, applications, network devices, and bare-metal servers. It can connect these deployed nodes to storage, add them to a load balancer, patch them for security, or perform any number of other operational tasks that are run by separate teams.

Ansible Automation Platform is a platform in your process pipeline for deploying infrastructure and connecting it, which simplifies the deployment and day-to-day management of your infrastructure. Consider the following components of your infrastructure that can be provisioned by Ansible:

- ▶ **Bare metal:** Underneath virtualization and cloud platforms is bare metal, and you still need to provision it depending on the situation. Ansible Automation Platform integrates with many data center management tools to both start and enact the provisioning steps that are required.
- ▶ **Virtualized:** From hypervisors to virtual storage and virtual networks, you can use Ansible Automation Platform to simplify the experience of cross-platform management. The large selection of available integrations gives you flexibility and the choice to manage your diverse environment.
- ▶ **Networks:** With Ansible network automation capabilities, users can configure, validate, and help ensure continuous compliance for physical network devices. Ansible Automation Platform can provision across multi-vendor environments, often replacing manual processes.
- ▶ **Storage:** Ansible Automation Platform can provision and manage storage in your infrastructure. Whether it is software-defined storage, cloud-based storage, or hardware storage appliances, you can find a module to benefit from the Ansible language.
- ▶ **Public cloud:** Ansible Automation Platform is packaged with hundreds of modules that support services on the largest public cloud platforms. Compute, storage, and networking modules enable playbooks to directly provision these services. Ansible can even act as an orchestrator of other provisioning tools.
- ▶ **Private cloud:** One way to deploy, configure, and orchestrate OpenStack private cloud is by using Ansible Automation Platform. You can use it to provision the underlying infrastructure, install services and applications, add computer hosts, and more.

### Centralizing your automation

Automation controllers help you scale IT automation, manage complex deployments, and speed productivity. Centralize and control your IT infrastructure with a visual dashboard, RBAC, job scheduling, integrated notifications, and graphical inventory management. You can use the automation controller REST API and CLI to embed the controller into existing tools and processes.

Figure 1-8 shows actions that can be improved by automation.

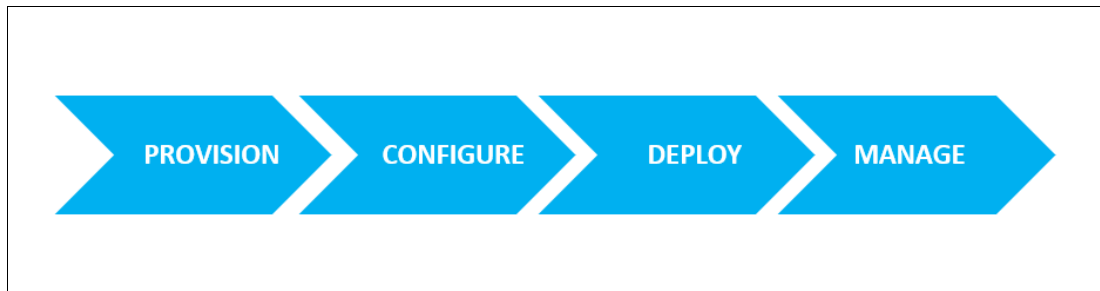


Figure 1-8 Automation does not stop with provisioning

By provisioning with Ansible Automation Platform, you can seamlessly migrate into configuration management, orchestration, and application deployment by using the same simple, human-readable automation language. For more information about provisioning with Ansible, see [Provisioning with Red Hat Ansible Automation Platform](#).

### 1.3.7 Patch management

One of the most critical activities with a direct relation to business continuity is the application of patches on your systems. If a critical patch is required to cover any potential breach, it is necessary to apply it as soon as possible while helping ensure that none of your systems go offline for too long a time, that is, affects your running services. There is a need for a reliable way of running an effective patch management strategy.

Ever wonder how you can apply patches on your systems, restart, and continue working with minimal downtime? Ansible can also function as a simple management tool to simplify patch management. Complicated administration tasks that take hours to complete can be managed with Ansible.

Although configuration management deals with maintaining the integrity and consistency of your system's components, patch management concentrates on updating and applying patches to these components. Through the usage of packaging modules and playbooks, Ansible effectively minimizes the amount of time that is required to patch your systems. Whenever you receive alerts for Common Vulnerabilities and Exposure (CVE) notifications or Information Assurance Vulnerability Alerts (IAVA), Ansible enables you to act swiftly in response to any potential dangers to your infrastructure.

### 1.3.8 Security and compliance

Security and compliance management is the ongoing process of monitoring and assessing systems to help ensure that they comply with industry and security standards, and corporate and regulatory policies and requirements.

Security and compliance management is important because noncompliance can result in fines, security breaches, loss of certification, or other damage to your business. Staying on top of compliance changes and updates prevents disruption of your business processes and saves money.

To successfully monitor and manage compliance for your business's infrastructure, you must achieve the following goals:

- ▶ **Assess:** Identify systems that are non-compliant, vulnerable, or unpatched.
- ▶ **Organize:** Prioritize remediation actions by effort, impact, and issue severity.
- ▶ **Remediate:** Quickly patch and reconfigure systems that require action.
- ▶ **Report:** Validate that changes were applied and report change results.

## **Security and compliance management challenges**

Here are a few things that can make security and compliance management difficult:

- ▶ **Changing security and compliance landscapes**  
Security threats and compliance changes evolve quickly, which require a rapid response to new threats and evolving regulations.
- ▶ **Distributed environments across multiple platforms**  
As infrastructures become more distributed across onsite and cloud platforms, it becomes more difficult to get a complete view of your environment and any risks and vulnerabilities that might be present.
- ▶ **Large environments and teams**  
Large, complex infrastructures and teams can complicate coordination across your environment and organization. In fact, system complexity can increase the cost of a data breach.

## **Security and compliance best practices and recommended tools**

The best way to meet each of these challenges is with a multifaceted approach that monitors all environments, identifies any regulatory inconsistencies, addresses those inconsistencies and bring them up to date and into compliance, and keep a record of these updates.

These best practices can help you stay abreast of any regulatory changes and keep your systems compliant:

1. **Regular system scans:** Daily monitoring can help you identify compliance issues and security vulnerabilities before they impact business operations or result in fees or delays.
2. **Deploy automation:** As the size of your infrastructure grows and changes, it becomes more challenging to manage manually. Using automation can streamline common tasks, improve consistency, and help ensure regular monitoring and reporting, which frees you to focus on other aspects of your business.
3. **Consistent patching and patch testing:** Keeping systems up to date can boost security, reliability, performance, and compliance. patches should be applied once a month to keep pace with important issues, and patching can be automated. patches for critical bugs and defects should be applied as soon as possible. Test patched systems for acceptance before placing them back into production.
4. **Connect your tools:** Distributed environments often contain different management tools for each platform. Integrate these tools through APIs so that you can use your preferred interfaces to perform tasks in other tools. Using a smaller number of interfaces streamlines operations and improves visibility into the security and compliance status of all systems in your environment.

Here are some of the security and compliance tools that can help:

- ▶ **Proactive scanning:** Automated scanning can help ensure that systems are monitored regularly and alert you to issues without expending too much staff time and effort.
- ▶ **Actionable insight:** Information that is tailored to your environment can help you quickly identify which compliance issues and security vulnerabilities are present, which systems are affected, and what potential impacts that you can expect.
- ▶ **Customizable results:** Define a business context to reduce false positives, manage business risk, and provide a more realistic view of your security and compliance status are ideal goals.
- ▶ **Prescriptive, prioritized remediation:** Prescriptive remediation instructions eliminate the need to research actions yourself, which save time and reduce the risk of mistakes. Prioritization of actions that is based on potential impact and systems that are affected help you make the most of limited patching windows.
- ▶ **Intuitive reporting:** Generating clear, intuitive reports about which systems are patched, which need to patch, and which are non-compliant with security and regulatory policies increases auditability and helps you gain a better understanding of the status of your environment.

### **Compliance and automation with IBM and Red Hat**

An automation strategy goes a long way to building capacity for checking systems for compliance without increasing time or cost. Manual compliance practices are more time-consuming, prone to human error, and harder to repeat or verify.

Selecting the correct automation technologies for your needs is key for rapid implementation across the data center and network software systems in hybrid environments.

#### ***Red Hat***

Red Hat has an end-to-end software stack for automation and management that includes the following products:

- ▶ Red Hat Enterprise Linux (RHEL)
- ▶ Red Hat Ansible Automation
- ▶ Red Hat Satellite
- ▶ Red Hat Insights

#### ***IBM***

IBM provides the following automation technologies:

- ▶ IBM QRadar®
- ▶ IBM PowerSC
- ▶ IBM Instana™®
- ▶ IBM Turbonomic®

## **1.3.9 Configuration management**

In today's IT environment, there are a growing number of servers and containers running to meet your business requirements. As the number of these instances increases, the amount of time spent managing and validating the configuration of those instances is also increasing. Ansible simplifies and automates configuration management, for example, by changing system parameters and devices. You can maintain a consistent setup across your enterprise, even in large environments with many systems.

An important aspect of configuration management is that when you run a playbook that you get the results that you expect, that is, the resultant configuration matches what is defined in the playbook. Ansible handles this task by helping ensure that playbooks are idempotent, which means that a Playbook runs again with the results being the same each time. This approach helps ensure that when you run a playbook to change a configuration parameter that the resulting configuration matches what is defined in the playbook.

### 1.3.10 Business continuity

Business continuity is an important aspect to keeping your infrastructure running to meet the demands of your users. Business continuity can be thought of a combination of technologies that provide a highly available (HA) environment, but it also involves the planning of your infrastructure to continue running in a disaster at the location of your primary data center, for example, a power outage, a fire, or a natural disaster that results in a flood or other damage. Helping ensure that your infrastructure is set up correctly to handle any hardware or software failures and site-wide outages that occur can be complex.

Ansible is a valuable tool for enhancing business continuity and disaster recovery (BCDR) efforts by automating various tasks and processes that are related to system recovery, data backup, and infrastructure provisioning. Using Ansible, new servers and instances can be created, either in the same site for HA or in another site for DR, which helps ensure that the infrastructure is ready when it is needed. In the event of increased demand during a disaster, Ansible helps to scale resources dynamically to handle the load and maintain business operations.

### 1.3.11 Application development

Modern business relies on applications, which are essential for your business and provide a competitive advantage. Fast and reliable application development is critical for success in a digital world.

CI/CD approaches can help you rapidly build, test, and deliver high-quality applications. CI/CD applies automation throughout the application lifecycle (from the integration and testing phases to delivery and deployment) to quickly produce tested and verified applications. It incorporates two different but related functions:

- ▶ Continuous integration (CI) helps developers rapidly verify functions and merge their code changes back to a shared branch more frequently. Merged code changes are validated by automatically building the application and running different levels of automated testing (typically unit and integration tests) to help ensure that the changes work. If testing discovers a conflict between new and existing code, CI makes it faster to patch those bugs.
- ▶ Continuous deployment (CD) automates the process of releasing an application to production. There are few manual gates in the development pipeline stage before production, so CD relies heavily on well-designed test automation. As a result, a developer's change to a cloud application could go live within minutes of writing it if it passes all automated tests. CD makes it much faster to continuously receive and incorporate user feedback. Together, CI and CD practices enable release changes to applications in smaller pieces, which make application deployment more reliable.

You can apply CI/CD to many components and assets within your organization, which includes applications, platforms, infrastructure, networking, and automation code. Automation is at the core of CI/CD pipelines. By definition, CI/CD pipelines require automation. Although it is possible to manually run each step in your development workflow, automation maximizes the value of your CI/CD pipeline. It helps ensure consistency across development, test, and production environments and processes so that you can build more reliable pipelines.

Ansible automates the major stages of CI/CD pipelines and therefore becomes the activating tool of DevOps methodologies. The automation technology that you choose can affect the effectiveness of your pipeline. Ideal automation technologies include these key features and capabilities:

- ▶ Ansible offers a simple solution for deploying applications. It provides the power to deploy multitier applications reliably and consistently, all from one common framework. You can configure key services and push application files from a single common system.
- ▶ Rather than writing custom code to automate your systems, your team writes simple task descriptions that even the newest team member can understand, which saves upfront costs and leads to a faster reaction to change over time.

With Ansible, you can write playbooks that describe the needed state of your systems, and then it does the hard work of getting your systems to that state. Playbooks make your installations, upgrades, and day-to-day management repeatable and reliable.

## 1.4 Introducing IBM Power

IBM Power is a family of midrange systems that can be used for mission-critical workloads by using hybrid multicloud technologies. IBM Power servers are high-performance, secure, and reliable servers built on the IBM POWER® processor architecture.

Figure 1-9 shows one of the newest systems (the IBM Power E1080) in the IBM Power product portfolio.



Figure 1-9 View of a Power E1080 node

IBM Power is built to be scalable and powerful while also providing flexible virtualization and management features. IBM Power supports many open-source tools, including Ansible.



Many of the most mission-critical enterprise workloads are run on IBM Power. The core of the global IT infrastructure, encompassing the financial, retail, government, health care, and every other sector in between, is composed of IBM Power servers, which are renowned for their industry-leading security, reliability, and performance attributes. For enterprise applications, including databases, application and web servers, ERP, and AI, many clients use IBM Power.

Enterprise IT delivery is changing as a result of digital transformation, and cloud computing is playing a key role. In consuming infrastructure, you need options and flexibility, and IBM Power is prepared for the cloud. Whether you use Kubernetes and Red Hat OpenShift to modernize enterprise applications; create a private cloud environment within your data center with adaptable pay-as-you-go services; use IBM Cloud to start applications as needed; or create a seamless hybrid management experience across your multicloud landscape, IBM Power delivers whatever hybrid multicloud approach that you choose.

The modern data center consists of a combination of on-premises and off-premises platforms, such as IBM Power, IBM Z, and x86. The applications range from monolithic to cloud-native, which are inherently some combination of bare metal, VMs, and containers. An effective hybrid cloud management solution must account for all these factors. IBM and Red Hat are positioned to accommodate the applications that you run today and the modernized applications of tomorrow.

### 1.4.1 IBM Power high availability

IBM Power delivers 99.999% availability with 25% less downtime than comparable offerings due to built-in recovery and self-healing functions for redundant components.<sup>7</sup> Organizations are also able to switch from an earlier Power server to the current generation while applications continue to run, which provides HA and minimal downtime when migrating.

### 1.4.2 IBM Power security

For the fourth straight year, IBM Power was rated as one of the most secure systems in 2022, with only 2.7 minutes or less of unplanned outages due to security issues.

- ▶ IBM Power is 2x more secure than comparable HPE Superdome servers.
- ▶ IBM Power is 6x compared to Cisco UCS servers.
- ▶ IBM Power is 16x compared to Dell PowerEdge servers.
- ▶ IBM Power is 20x compared to Oracle x86 servers.
- ▶ IBM Power is more than 60x compared to unbranded, white box servers.

Security breaches were also detected immediately or within the first 10 minutes in 95% of the IBM Power servers that were surveyed, which results in better chances that a business suffers little to no downtime or is susceptible to damaged, compromised, or stolen data.<sup>8</sup>

---

<sup>7</sup> Source: <https://itic-corp.com/itic-2023-reliability-survey-ibm-z-results/>

<sup>8</sup> Source:

<https://itic-corp.com/ibm-z-ibm-power-systems-lenovo-thinksystem-servers-most-secure-toughest-to-crack/>

### 1.4.3 IBM Power, operational efficiency, and sustainability

A recent user case study illustrated how IBM Power enabled a customer to increase user application performance by 20%, and they met their sustainability goals.<sup>9</sup> Moving to IBM Power and IBM FlashSystem® storage meant that the customer was able to leverage their SAP S/4HANA operations and meet their climate objectives.

### 1.4.4 Streamlining AI operations with advanced on-chip technologies

IBM Power servers deliver 5X faster AI inferencing per socket for high precision math over the previous generation.<sup>10</sup> This feat is accomplished through multiple Matrix Math Accelerator (MMA) units in each POWER processor core. MMAs allow IBM Power servers to forgo external accelerators, such as GPUs and related device management, when running machine learning and inferencing workloads.

At the time of writing, IBM Power servers range from scale-out servers that start with 4 cores and 32 GB of memory on the IBM Power S1014 to enterprise systems with up to 240 cores and 64 TB of memory on the IBM Power E1080.

**Note:** For more information about the full line of IBM Power server models, see [IBM Power](#).

### 1.4.5 POWER processors and architecture

The POWER processor is a family of 64-bit superscalar, simultaneous multithreading, and multi-core microprocessors that are designed and sold by IBM. POWER processors are used for IBM Power servers, the Hardware Management Console (HMC), and in IBM storage solutions, such as the IBM DS8900F and the IBM Converged Archive Solution.

As an architecture, Power based processors have been used in various applications, such as network routers, workstations, game consoles, and even on the Curiosity and Perseverance rovers on Mars.

As AI infrastructure challenges increase due to more AI models being deployed in production, IBM Power addresses these challenges with in-core AI inferencing and machine learning through its built-in MMA. You gain the capability of “AI at the point data”, where you can perform AI inferencing without needing to ingest your data from an external source. This approach provides AI operations with a significant performance boost.

The Power based processor also provides security within the system itself through Transparent Memory Encryption, where data is encrypted by cryptography engines that are in the processor core, where memory is located. The processor gains four times the speed than average encryption.

Power also features reliability, availability, and serviceability (RAS) capabilities, such as advanced recovery, diagnostics, and Open Memory Interface (OMI) attached advanced memory DIMMs that deliver 2X better reliability and availability than industry-standard DIMMs.

The latest version of POWER processors is the Power10, built on a 7-nm design that is 50% faster than its predecessor and 33% more energy efficient.

<sup>9</sup> Source: <https://www.ibm.com/case-studies/mondi-group-systems-hardware-sap-s4-hana>

<sup>10</sup> Source: <https://www.ibm.com/it-infrastructure/resources/power-performance/e1080/>

Power10 chip benefits are the result of important evolutions of many of the components that were in previous IBM POWER chips. Several of these important Power10 processor improvements are listed in Table 1-3.

Table 1-3 Power10 processor chip technology

Technology	Power10 processor chip
Processors die size	602 mm <sup>2</sup>
Fabrication technology	<ul style="list-style-type: none"> <li>▶ CMOS 7-nm lithography</li> <li>▶ 18 layers of metal</li> </ul>
Maximum processor cores per chip	15
Maximum execution threads per core / chip	8 / 120
Maximum L2 cache core	2 MB
Maximum On-chip L3 cache per core / chip	8 MB / 120 MB
Number of transistors	18 billion
Processor compatibility modes	Support for Power Instruction Set Architecture (ISA) of IBM Power8® and IBM Power9®

Figure 1-10 shows the Power10 processor die with several functional units that are labeled. Sixteen SMT8 processor cores are shown, but the dual-chip module (DCM) with two Power10 processors provide 12-, 18-, or 24-core for Power E1050 server configurations.

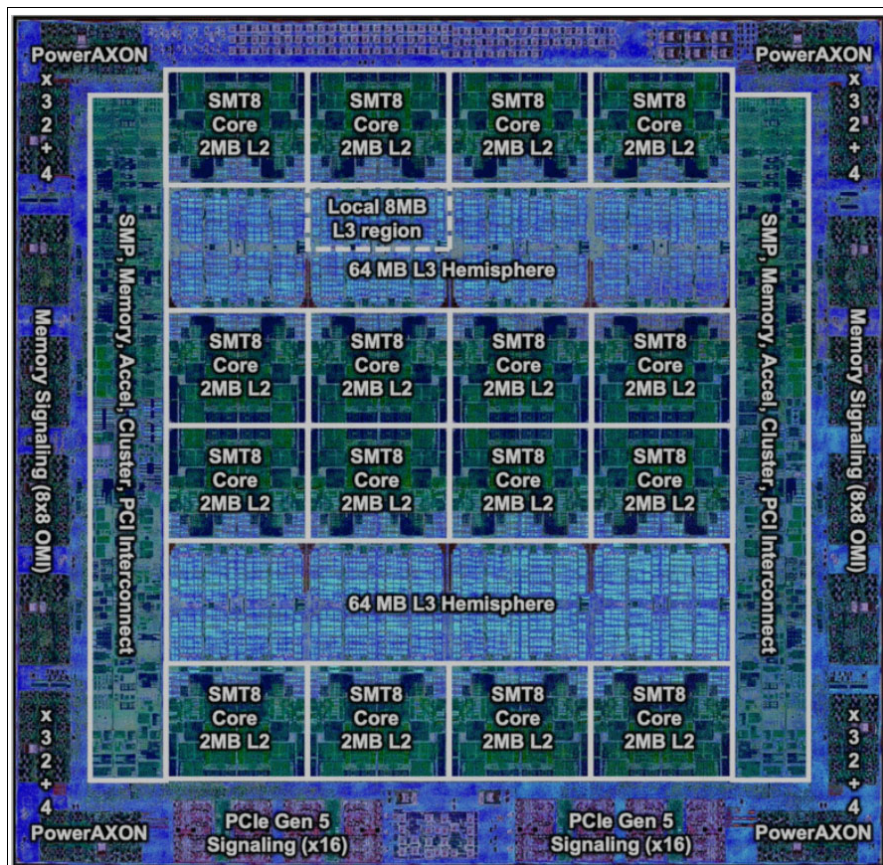


Figure 1-10 The Power10 processor chip (die photo courtesy of Samsung Foundry)

## 1.4.6 PowerVM and virtualization

IBM PowerVM is a virtualization environment feature that is available on IBM Power that support VMs by enabling the creation of micro-partitions (also known as logical partitions (LPARs)). With PowerVM, you can consolidate VMs that run multiple workloads onto fewer systems, which result in reduced costs, increased efficiency, better return on investment, faster deployment, workload security, and better server utilization. PowerVM enables a Power server to have up to 1000 VMs on a single server running a mix of various operating systems (OSs) and environments simultaneously.

PowerVM also provides other IBM Power advanced features:

- ▶ IBM Micro-Partitioning®

Enables a partition or VM to initially occupy as small as 0.05 processing units, or one-twentieth of a single processor core, and allows adjustments as small as a 100th (0.001) of a processor core. This approach provides tremendous flexibility in adjusting your resources according to the exact needs of your workload.

- ▶ Shared Processor Pools

Enables effective overall utilization of system resources by automatically applying only the required amount of processor resource that is needed by each partition. The hypervisor can automatically and continually adjust the amount of processing capacity that is allocated to each partition or VM based on system demand. You can set a shared processor partition so that if a VM requires more processing capacity than its assigned number of processing units, the VM can use unused processing units from the shared processor pool.

- ▶ Virtual I/O Server (VIOS)

VIOS shares storage and network resources across several VMs simultaneously, which helps avoid excessive costs by configuring the precise amount of hardware resources that is needed by the system.

- ▶ Live Partition Mobility (LPM)

LPM moves running VMs across different physical systems without disrupting the OS and applications running within them.

- ▶ Share Storage Pool (SSP)

SSP provides distributed storage resources to VIOSs in a cluster.

- ▶ Dynamic LPAR (DLPAR) operations

Dynamically allocate more resources, such as available cores and memory to a VM without stopping the application.

- ▶ Performance and Capacity Monitoring

Supports the gathering of important statistics to provide an administrator with information regarding physical resource distribution among VMs and continuous monitoring of resource utilization levels, which helps to help ensure that the resources are evenly distributed and optimally used.

- ▶ Remote Restart

Provides quick recovery in your environment by restarting a VM on a different physical server when an error causes an outage.

**Note:** For more information and a comprehensive description of PowerVM and its capabilities, see *Introduction to IBM PowerVM*, SG24-8535.

## 1.4.7 Supported operating systems

At the time of writing, Power10 processor-based systems are supported by the OS versions that are shown in Table 1-4.

Table 1-4 Power10 operating system support

Operating system	Supported versions
AIX	7.3 TL0 or later 7.2 TL4 or later (with any I/O configuration) 7.1 TL5 or later (through VIOS only)
IBM i	7.3 TR12 or later (Various levels of 7.3, 7.4, and 7.5 are supported.)
PowerVM VIOS	4.1.0.10 or later 3.1.4.10 or later 3.1.3.21 or later 3.1.2.40 or later
Red Hat OpenShift Container Platform	4.9 or later
RHEL	9.0 or later 8.4 or later
SUSE Linux Enterprise Server	15.3 or later
Ubuntu	22.04 or later

**Note:** Software maps detailing which OS versions are supported on which specific IBM Power server models (including previous generations of IBM Power) can be found in these [IBM Support pages](#).

## 1.4.8 Key benefits of IBM Power compared to x86 servers

Often, the misapprehension surrounding standardization on x86 is that it is the platform with lower acquisition costs. However, these costs are often at the expense of performance, scalability, reliability, and manageability. The return on investment and total cost of ownership of x86 servers also do not match IBM Power servers. IBM delivers the better overall platform compared to x86 due to the following benefits of IBM Power:

- ▶ The world record SAP SD-two tier benchmark results with 8 sockets (120 cores), which beats the best 16 socket (448 cores) result from the x86 platform.<sup>11</sup>
- ▶ Power delivers a per-core performance that is 2.5X faster than Intel Xeon Platinum, which is a world record 8-socket single server result on the SPEC CPU 2017 benchmark.<sup>12</sup>
- ▶ When running containerized applications and databases on an IBM Power E1080 compared to running the same workloads on an x86 server, IBM Power delivers a 48% lower 3-year TCO, 4.3X more throughput per core, and 4.1X better price-performance. You can run the same number of workloads with fewer servers with four times less the footprint, four times fewer software licenses, and four times the energy savings.<sup>13</sup>

<sup>11</sup> Source: <https://www.sap.com/about/benchmark.html>

<sup>12</sup> Source: <https://www.spec.org/cpu2017/results/>

<sup>13</sup> Source: <https://www.ibm.com/it-infrastructure/resources/power-performance/e1080/#5>

- ▶ For the 14th straight year, IBM Power delivers the top reliability results, which are better than any Intel x86 platform, and only exceeded by IBM Z. IBM Power also reported fewer data breaches (one) in the same period compared to x86 platforms.<sup>14</sup>
- ▶ As shown by the list of supported OSs in Table 1-4 on page 29, IBM Power can run many AIX, IBM i, or Linux workloads simultaneously, which provides flexibility in virtualization that is unmatched by any x86 offering.

Both Power and x86 architectures are established, mature foundations for modern workloads. But Power stands out for its efficiency, deeply integrated virtualization, highly dependable availability and reliability, and its unparalleled capability of supporting enterprise-class workloads that do not require the massive infrastructure that you would need to do the same with x86 hardware.

## 1.5 Ansible for Power

This section describes Ansible clients across both on-premises IBM Power environments and IBM Cloud Power Systems Virtual Server. Ansible can manage many clients running on IBM Power, including AIX, IBM i, Linux, HMC, VIOS, and Red Hat OpenShift Container Platform. Section 3.4, “Preparing your systems to be Ansible clients” on page 149 describes how to set up your VMs as Ansible clients. In addition to the generic modules and collections that Ansible provides, IBM has developed many collections specifically for IBM Power servers. These collections can be found among Ansible Galaxy for community-supported content, or Red Hat Automation Hub for certified content.

### 1.5.1 IBM Power Collections on Ansible Galaxy

Ansible Galaxy is a source for community-based Ansible content, which holds reusable collections from thousands of contributors, including IBM. These collections include modules, plug-ins, playbooks, and roles that anyone can download and use.

Table 1-5 shows the IBM Power collections that were developed by IBM.

Table 1-5 Links to IBM Power collections

Collection	URL
Power AIX	<a href="https://galaxy.ansible.com/ui/repo/published/ibm/power_aix/">https://galaxy.ansible.com/ui/repo/published/ibm/power_aix/</a>
Power IBM i	<a href="https://galaxy.ansible.com/ui/repo/published/ibm/power_ibmi/">https://galaxy.ansible.com/ui/repo/published/ibm/power_ibmi/</a>
Power HMC	<a href="https://galaxy.ansible.com/ui/repo/published/ibm/power_hmc/">https://galaxy.ansible.com/ui/repo/published/ibm/power_hmc/</a>
Power VIOS	<a href="https://galaxy.ansible.com/ui/repo/published/ibm/power_vios/">https://galaxy.ansible.com/ui/repo/published/ibm/power_vios/</a>
PowerVC	Ansible support for PowerVC is provided by using the Ansible Collection for OpenStack.

<sup>14</sup> Source: <https://itic-corp.com/itic-2022-global-server-reliability-results/>

## 1.5.2 IBM Power Collections on Red Hat Automation Hub

These same collections can also be downloaded from [Red Hat Automation Hub](#). The Ansible Automation Hub features Red Hat Ansible Certified Content, which is prebuilt, fully supported, and certified automation content from Red Hat and 60+ partners, which include Cisco, Microsoft, CyberArk, Dynatrace, and ServiceNow. It also includes validated content, such as prebuilt playbooks and roles that you can use, customize, and learn from.

Figure 1-11 shows the Red Hat Automation Hub interface showing these collections.

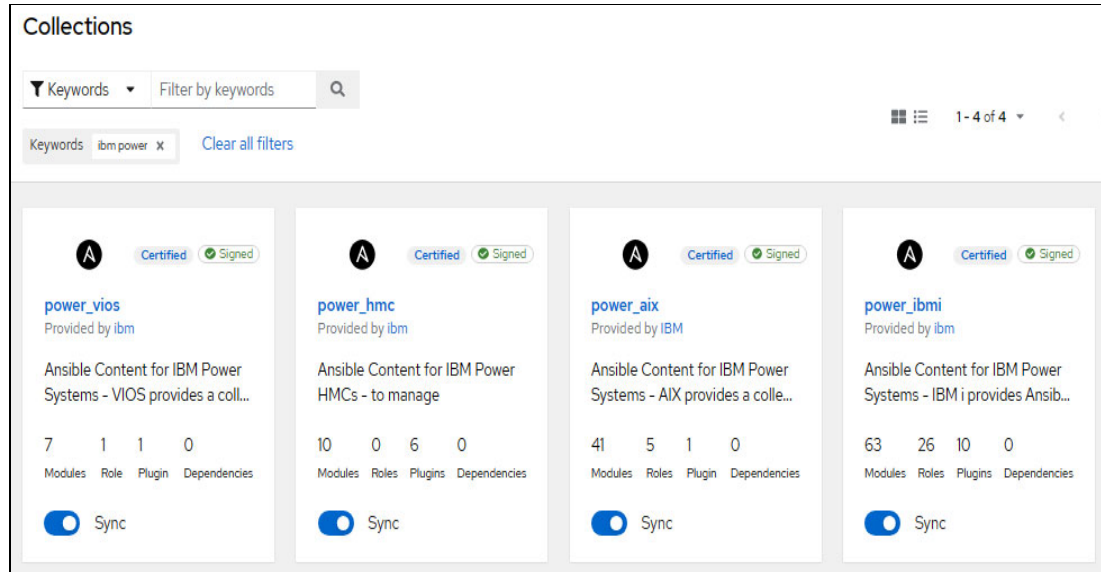


Figure 1-11 Collection on the Red Hat Automation Hub

The following sections describe the benefit of using Ansible across these different environments, and provide more information about the specific collection contents.

## 1.5.3 Ansible for Linux on Power

This section describes the following topics:

- ▶ Introduction
- ▶ SUSE Linux Enterprise Server
- ▶ Getting started with Linux management
- ▶ Login options
- ▶ A simple Ansible playbook
- ▶ Updating packages
- ▶ Cross-platform playbook considerations

### Introduction

A major benefit of Linux is that it is open source. The software is unencumbered by licensing fees and its source code is freely available. Many Linux distributions are available for almost every computing platform.

Here are the supported Linux distributions on IBM Power servers:

- ▶ RHEL
- ▶ SUSE Linux Enterprise Server
- ▶ Ubuntu (support is available from Canonical)
- ▶ Red Hat OpenShift Container Platform

### ***Red Hat Enterprise Linux***

RHEL is the most-deployed commercial Linux distribution in the public cloud.<sup>15</sup> RHEL is an open-source OS that provides a foundation to scale existing applications and deploy emerging technologies across the following environments:

- ▶ Bare metal
- ▶ Virtual
- ▶ Container
- ▶ Cloud

### ***SUSE Linux Enterprise Server***

SUSE Linux Enterprise Server is an enterprise Linux distribution that is enabled for the cloud, which also has a strong presence on the IBM Power platform.

### ***Ubuntu***

IBM and Canonical collaborated to make the Ubuntu distribution on IBM Power. The availability of Ubuntu strengthens the Linux OS choices that are available on IBM Power technology. Support for Ubuntu is available directly from Canonical.

### ***Red Hat OpenShift Container Platform***

Red Hat OpenShift Container Platform is a consistent hybrid cloud foundation for building and scaling containerized applications. Red Hat OpenShift is trusted by thousands of customers in every industry to deliver business-critical applications, whether they are migrating existing workloads to the cloud or building new experiences for customers. It is also backed by one of the leading Kubernetes contributors, Red Hat.

Red Hat OpenShift Container Platform runs on Red Hat CoreOS, which represents the next generation of single-purpose container OS technology by providing the quality standards of RHEL with automated, remote upgrade features. Red Hat CoreOS is the only supported OS for Red Hat OpenShift Container Platform control plane (master) nodes. Although Red Hat CoreOS is the default OS for all cluster machines, you can create compute (worker) nodes that use RHEL as their OS.

Ansible supports the Red Hat OpenShift Container Platform as either a control node or a client node.

## **Getting started with Linux management**

Managing Linux on IBM Power with Ansible does not require optional Ansible modules or collections. It is served by the existing `ansible-core` modules that are included with all Ansible installations (`ansible.builtin.*`) and common community modules (`ansible.community.*`), some of which are shown in Table 1-6 on page 33.

---

<sup>15</sup> [Management Insight Technologies. "The State of Linux in the Public Cloud for Enterprises," February 2018.](#)



Table 1-6 Ansible modules commonly used with Linux targets

Ansible module name	Fully qualified collection name	Description
apt	ansible.builtin.apt	Manages apt packages for Ubuntu and Debian distributions.
blockinfile	ansible.builtin.blockinfile	This module inserts, updates, or removes a block of multi-line text that is surrounded by customizable marker lines.
command	ansible.builtin.command	Runs commands on targets.
copy	ansible.builtin.copy	Copies files to remote locations.
debug	ansible.builtin.debug	Prints statements during execution.
dnf	ansible.builtin.dnf	Manages packages with the dnf package manager for Python 3.
fetch	ansible.builtin.fetch	Fetches files from remote nodes.
file	ansible.builtin.file	Manages files and file properties.
lineinfile	ansible.builtin.lineinfile	Manages lines in text files.
template	ansible.builtin.template	Templates a file out to a target host by using jinja2.
user	ansible.builtin.user	Manages user accounts.
yum	ansible.builtin.yum	Manages packages with the Yellowdog Updater, Modified (YUM) package manager. Python 2 only.
zypper	ansible.general.zypper	Manages packages on SUSE.

**Note:** Table 1-6 on page 33 represents a small selection of Ansible modules that are available. For more information and a comprehensive list of Ansible modules and collections, see [this document](#).

Ansible accesses Linux hosts in the inventory through SSH from the Ansible Controller node, with authentication through either password, or public and private key pairs. To learn how to choose a controller node and install Ansible, see Chapter 3, “Getting started with Ansible” on page 103.

No special modules or collections are required to access a Linux host. For example, to access a Linux host that named `sles15sp5` from a controller node by using a username and password, use the example that is shown in Example 1-1.

*Example 1-1 Accessing a Linux host by using a password*

---

```
$ ansible -m ping sles15sp5 --ask-pass --user ansible
SSH password:
sles15sp5 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.6"
  },
  "changed": false,
  "ping": "pong"
}
```

---

To set up password-less authentication by using SSH public and private keys, use the `ssh-copy-id` command, as shown in Example 1-2, to copy a preexisting public key to a Linux host.

*Example 1-2 Configuring a password-less login by using the ssh-copy-id command*

---

```
$ ssh-copy-id ansible@sles15sp5
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new keys to filter out
any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 keys remain to be installed -- if you are prompted
now it is to install the new keys
(ansible@sles15sp5) Password:
```

Number of keys added: 1

Now, try logging in to the machine with: `"ssh 'ansible@sles15sp5'"` and check to make sure that only the keys you wanted were added.

```
$ ssh ansible@sles15sp5
Last login: Mon Aug 7 16:36:31 2023 from 192.168.115.249
ansible@sles15sp5:~>
```

---

Once password-less logins to your Linux hosts are configured, access through Ansible becomes simpler, as shown in Example 1-3.

*Example 1-3 Using a passwordless login*

---

```
$ ansible -m ping sles15sp5 --user ansible
sles15sp5 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.6"
  },
  "changed": false,
  "ping": "pong"
}
```

---

Your Ansible command can be further simplified by specifying `<ansible_user>` in either the inventory or a `ansible.cfg` file.

## Login options

Many actions in Ansible require superuser or root access. There are several options to acquire authorization for those operations.

### Using the root user ID

Direct access to a Linux host as the root user is possible, and can be permitted or denied by the `PermitRootLogin` setting in `/etc/ssh/sshd_config` on the target host.

Here are the valid options for `PermitRootLogin`:

- ▶ `yes`
- ▶ `no`
- ▶ `prohibit-password`
- ▶ `without-password`
- ▶ `forced-commands-only`

A value of `yes` permits root logins by using a password, and a setting of `no` denies access. The settings `prohibit-password` and `without-password` permit root logins, but only by an SSH key-pair, and not a password. The option `forced-commands-only` permits password-less login by root, but only to run predefined commands.

Your organization's security policy may not permit direct logins as the root user through SSH. If so, log in as a regular user and then use the `sudo` or `su` command to perform tasks that requires superuser access. (This process is called privilege escalation.)

### Using sudo

To use `sudo`, the user issuing the `sudo` command must be listed in the `/etc/sudoers` file on the target host. The OS groups `sudo` or `wheel` are commonly used for this type of authorization. An example of a `/etc/sudoers` file enabling the members of the `sudo` group to run any command as root is shown in Example 1-4.

*Example 1-4 Sample of `/etc/sudoers` file enabling root access for members of the `sudo` group*

---

```
# Allow members of group sudo to run any command
%sudo ALL=(ALL:ALL) ALL
```

---

### Using su

The `su` command is another way to gain superuser privileges. In this case, the user provides the root password to obtain a privilege escalation rather than their own password. The usage of `sudo` over `su` is preferred due to the greater granularity of control by using `sudo`.

### Privilege escalation

To use privilege escalation in Ansible, use the `become` keyword. You can use the `become` keyword in a playbook, an `ansible.cfg` file, or on the CLI.

**Note:** Privilege escalation by using `become` is not limited to the root user. You can specify another user with `become` with the `become_user` option. For example, you might use `become_user: apache` to perform tasks as the web server owner.

## A simple Ansible playbook

A common task in Ansible is to install an OS package, for example, a tool like `tcpdump`.

RHEL, SUSE Linux Enterprise Server, and Ubuntu all use their own separate package management systems, and have separate Ansible modules to install packages:

- ▶ RHEL: `ansible.builtin.yum`
- ▶ SUSE Linux Enterprise Server: `community.general.zypper`
- ▶ Ubuntu: `ansible.builtin.apt`

However, there is also a generic package management module that is named `ansible.builtin.package` that can be used to make your playbook more portable across Linux distributions.

Here is a simple example playbook that performs the following tasks:

- ▶ Run the playbook on the hosts in the `power-linux` inventory group.
- ▶ Log in to the Linux hosts as the user `ansible`.
- ▶ Use `sudo` to become the root user.
- ▶ Install the `tcpdump` package by using the generic `ansible.builtin.package` module.
- ▶ The package module helps ensure that the package that is installed if not already present by using the `in state` keyword with the value of `present`.

The playbook is shown in Example 1-5.

*Example 1-5 Simple playbook `install_pkg.yaml`*

---

```
---
- hosts: power-linux
  remote_user: ansible
  become: true

  tasks:
    - name: install tcpdump package
      ansible.builtin.package:
        name: tcpdump
        state: present
```

---

The inventory file contains the host definitions that are shown in Example 1-6.

*Example 1-6 Inventory example for the example playbook*

---

```
[power-linux]
rhe17ppc64
rhe18ppc64
rhe19ppc64
sles15ppc64
ubuntu20ppc64
ubuntu22ppc64
```

---

To run the playbook, run the command that is shown in Example 1-7.

*Example 1-7 Running the `ansible-playbook` command*

---

```
ansible-playbook --ask-pass --ask-become-pass --inventory inventory
install_pkg.yaml
```

---

The output of the `ansible-playbook` command is shown in Example 1-8.

*Example 1-8 Output from running the playbook*

---

```
$ ansible-playbook --ask-pass --ask-become-pass --inventory inventory install-pkg.yaml
SSH password:
BECOME password[defaults to SSH password]:

PLAY [power-linux]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [rhe17ppc64]
ok: [ubuntu22ppc64]
ok: [rhe18ppc64]
ok: [rhe19ppc64]
ok: [ubuntu20ppc64]
ok: [sles15ppc64]

TASK [install package]
*****
*****
ok: [ubuntu20ppc64]
changed: [ubuntu22ppc64]
ok: [rhe17ppc64]
ok: [rhe19ppc64]
changed: [rhe18ppc64]
changed: [sles15ppc64]

PLAY RECAP
*****
*****
rhe17ppc64      : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
rhe18ppc64      : ok=2   changed=1   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
rhe19ppc64      : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
sles15ppc64     : ok=2   changed=1   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu20ppc64   : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu22ppc64   : ok=2   changed=1   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu20ppc64   : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
ubuntu20ppc64   : ok=2   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0
```

---

The output from the command in Example 1-8 shows that only the target hosts `rhe18ppc64`, `sles15ppc64`, and `ubuntu22ppc64` have a status value of `changed`, and therefore were the only hosts that required the `tcpdump` package to be installed.

You can further simplify the command by putting options in an `ansible.cfg` file rather than specifying them each time that you run a playbook.

An example `ansible.cfg` file that you can use when running a playbook as a non-root user, and prompting for the user and sudo password is shown in Example 1-9.

*Example 1-9 Sample ansible.cfg file for playbook that uses become for privilege escalation*

---

```
[defaults]
inventory = inventory
remote_user = ansible
ask_pass = true

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = true
```

---

An example `ansible.cfg` file that you can use when running a playbook to use SSH keys to log in to target hosts directly as root is shown in Example 1-10.

*Example 1-10 Sample ansible.cfg file for password-less logins as the root user*

---

```
[defaults]
inventory = inventory
remote_user = root
ask_pass = false
```

---

## Updating packages

A common use for Ansible playbooks is to help ensure compliance by keeping packages updated with the latest security patches.

### **Generic package module**

The generic `ansible.builtin.package` module that is used in the example playbook that is shown in Example 1-5 on page 36 can also be used to update all packages, as shown in Example 1-11.

*Example 1-11 Using a generic package module to update all packages to the latest level*

---

```
- name: Update all packages to the latest available
  ansible.builtin.package:
    name: '*'
    state: latest
```

---

This generic package module is suitable for simple playbooks, but sometimes you need more fine-grained control of the package updates. The next section introduces the `ansible.builtin.dnf` module, which provides more capabilities.

### **Using the dnf module on Red Hat Enterprise Linux**

There is an `ansible.builtin.yum` module, but it is written for Python 2. For newer deployments, you should use `ansible.builtin.dnf` instead. The `dnf` module allows more control of the installation, upgrade, and removal of packages than the generic package module.

For example, you want to upgrade a list of packages, but only if they are already installed. The task code to do this task looks like Example 1-12.

*Example 1-12 Upgrading a list of packages but not installing them if they are not present*

---

```
- name: Update required packages
  dnf:
    name:
      - firefox
      - curl
      - python3
    update_only: yes
    state: latest
```

---

Full documentation for the dnf module can be found in the [dnf collection description](#).

### **Using the zypper module on SUSE Linux Enterprise Server**

If you are running SUSE Linux Enterprise Server, there is a zypper module that is similar to the dnf module for RHEL. The zypper module for SUSE Linux Enterprise Server is part of the `community.general` collection, and must be installed before use. Use the `ansible-galaxy` command to install the collection, as shown in Example 1-13.

*Example 1-13 Installing the community.general collection by using ansible-galaxy*

---

```
ansible-galaxy collection install community.general
```

---

After the zypper module is installed, its options are similar to the dnf module. For more information about the zypper module, see [Ansible Community Documentation](#).

### **Using the apt module on Ubuntu**

The apt module options are similar to the dnf module. One unique option for the apt module is `cache_valid_time`, which prevents the updating of repository caches until the age of the cache exceeds the value of `cache_valid_time`.

The full documentation for the apt module can be found at [Ansible Community Documentation](#).

## **Cross-platform playbook considerations**

If you want to write a playbook that runs on Linux hosts with different architectures, some packages are not present across all platforms.

For example, the Ubuntu packages `intel-microcode` and `amd64-microcode` are not present on ppc64 platforms. If you wanted to explicitly update these packages to the latest version in your playbook, use the tasks in Example 1-14 to update the packages on x86\_64 platforms, which do not generate an error on ppc64 platforms.

*Example 1-14 Only update packages if they are found in 'ansible\_facts'*

---

```
- name: Update intel-microcode if present (that is, not on ppc64)
  package:
    name: intel-microcode
    state: latest
    when: "'intel-microcode' in ansible_facts.packages"

- name: Update amd64-microcode if present (that is, not on ppc64)
  package:
```

```
name: amd64-microcode
state: latest
when: "'amd64-microcode' in ansible_facts.packages"
```

---

Conversely, packages from the Linux on IBM Power repository are not present on other architectures. For more information, see [IBM Linux on Power Tools](#).

## 1.5.4 Ansible for AIX

Here are the supported AIX versions on IBM Power:

- ▶ AIX 7.1
- ▶ AIX 7.2
- ▶ AIX 7.3

Ansible accesses AIX hosts in the inventory through SSH, with authentication through either a password, or public and private key pairs.

All nodes must be enabled to run open-source packages and have Python 3 installed. Beginning with AIX 7.3, Python 3 is automatically preinstalled with the OS. On the controlling node, also Ansible 2.9 or later must be installed.

All packages can be downloaded from [AIX Toolbox for Linux Applications](#). As a best practice, use the [DNF package](#).

This script downloads `rpm.rte`, `dnf_bundle.tar`, and `rpm.rte-4.13.0.x`, which are prerequisites for `.`. The `dnf_bundle.tar` file contains `.` and its dependent packages. This script checks whether any packages from `dnf_bundle` are already installed, and then installs the packages.

No special modules or collections are required to access an AIX host. For example, to access an AIX host that is named `aix7.3` by using a username and password, it is as simple as shown in Example 1-15.

*Example 1-15 Accessing an AIX host by using a password*

---

```
$ ansible -m ping aix7.3 --ask-pass --user ansible
SSH password:
aix7.3 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

---

Alternatively, to set up password-less authentication by using SSH public and private keys, use the `ssh-copy-id` command, as shown in Example 1-2 on page 34 to copy a preexisting public key to a AIX host.

Once the key exchange is done, you can include the private key in your node controller inventory file and run the command directly without using a password, as shown in Example 1-16.

*Example 1-16 Running the command without a password*

---

```
$ ansible aix -b -m shell -a "bootinfo -s hdisk0"
aix7.3 | CHANGED => | rc=0 >>
51200
```

---



## Ansible community for IBM AIX

There are several options for obtaining Ansible content for IBM AIX:

- ▶ Galaxy  
The [Ansible Galaxy URL](#) Ansible Content for IBM Power Systems - AIX provides a collection of content that is used to manage and deploy AIX on Power servers.
- ▶ GitHub repository  
This [GitHub repository](#) provides modules that can be used to manage configurations and deployments of AIX on Power servers. The collection content helps to include workloads on Power platforms as part of an enterprise automation strategy through the Ansible ecosystem.
- ▶ Documentation  
The dedicated [documentation portal](#) offers detailed insights into the usage, configuration, and best practices of Ansible for AIX. This resource guides users through the process of integrating Ansible into their AIX workflows.
- ▶ Automation Hub  
[The Automation Hub](#) offers Red Hat Ansible Certified Content for IBM Power that helps you manage workloads on the Power platform as part of your wider enterprise automation strategy through the Ansible Automation Platform ecosystem. The collection is named [ibm.power\\_aix](#).

## IBM Power AIX collection

The IBM Power AIX collection provides modules that can be used to manage configurations and deployments of AIX on Power servers. The collection content helps to include workloads on Power platforms as part of an enterprise automation strategy through the Ansible ecosystem.

### ***AIX specific Ansible modules***

There are many AIX specific modules that are found at [Ansible Content for IBM Power - AIX](#) on the Ansible site, some of which are shown in Table 1-7. These modules offer functions that are targeted for AIX environments.

Table 1-7 *Specific Ansible modules for AIX*

Module	Minimum Ansible version	Description
<code>_nim_upgradeios</code>	2.9	Use a Network Installation Manager (NIM) to update a single or a pair of VIOSs.
<code>aixpert</code>	2.9	System security settings management.
<code>alt_disk</code>	2.9	Alternative rootvg disk management.
<code>backup</code>	2.9	Data or system volume group backup management.
<code>bootlist</code>	2.9	Alters the list of boot devices that are available to the system.
<code>bosboot</code>	2.9	Creates a boot image.
<code>chsec</code>	2.9	Modifies AIX stanza files.
<code>devices</code>	2.9	Devices management.
<code>emgr</code>	2.9	System interim fixes management.

Module	Minimum Ansible version	Description
filesystem	2.9	Local and NFS file systems management.
flrtvc	2.9	Generates an FLRTVC report, and downloads and installs security and HIPER fixes.
geninstall	2.9	Generic installer for various packaging formats.
group	2.9	Manages presence, attributes, and members of AIX groups.
inittab	2.9	Manages inittab entries on AIX.
installp	2.9	Installs and updates software.
internal.nim_select_target_disk	2.9	Verifies or autoselects a disk that is used for an alternative disk migration role.
lpar_facts	2.9	Reports LPAR-related information as facts.
lpp_facts	2.9	Returns installed software products or fixes as facts.
lvg	2.9	Configures AIX logical volume manager (LVM) volume groups.
lvm_facts	2.9	Reports LVM information as facts.
lvol	2.9	Configures AIX LVM logical volumes.
mkfilt	2.9	Activates or deactivates the filter rules.
mktcpip	2.9	Sets the required values for starting TCP/IP on a host.
mktun	2.9	Creates, activates, deactivates, and removes tunnels.
mount	2.9	Mounts or unmounts a file system or device on AIX.
mpio	2.9	Returns information about MultiPath I/O capable devices.
nim	2.9	Performs NIM operations, such as server setup, installing packages, and updating SPs or TLs.
nim_backup	2.9	Uses NIM to manage the backup of LPAR or VIOS clients.
nim_flrtvc	2.9	Uses NIM to generate an FLRTVC report, and downloads and installs security and HIPER fixes.
nim_resource	2.9	Shows, defines, or deletes NIM resource objects.
nim_suma	q	Uses NIM to download fixes, SPs, or TLs from the IBM Patch Central website.
nim_updateios	2.9	Uses NIM to update a single or a pair of VIOSs.

Module	Minimum Ansible version	Description
nim_vios_alt_disk	2.9	Uses NIM to create or clean up an alternative rootvg disk of VIOS clients.
nim_vios_hc	2.9	Checks whether a pair of VIOSs can be updated.
nim_viosupgrade	2.9	Uses NIM to upgrade VIOSs with the viosupgrade tool from a NIM master.
reboot	2.9	Restarts AIX machines.
smtctl	2.9	Enables and disables Simultaneous MultiThreading Mode.
suma	2.9	Downloads or installs fixes, SPs, or TLs from the IBM Patch Central website.
tunables	2.9	Modifies, resets, or shows tunables for various components on AIX.
tunfile_mgmt	2.9	Saves, restores, validate, or modifies the tunables configuration file for various components on AIX.
user	2.9	Creates users or changes or removes attributes of users on AIX.

### ***Specific Ansible roles for AIX***

Table 1-8 shows the Ansible roles that run specific tasks to configure an AIX host. These tasks include activities such as configuring TCP/IP services on the AIX host or migrating to a higher AIX level by using an alternative disk.

*Table 1-8 Specific Ansible roles for AIX*

Ansible role	Minimum Ansible version	Description
bootptab	2.9	Ansible role for modifying the bootptab file on AIX.
inetd	2.9	Ansible role for enabling or disabling InetD services on AIX.
nim_alt_disk_migration	2.9	Ansible role for migrating an alternative disk to a higher AIX level.
power_aix_bootstrap	2.9	Ansible role for bootstrapping IBM Power AIX servers with DNF package managers.
power_aix_vioshc	2.9	Ansible role for installing vioshc on a NIM master.

### ***Ansible extensions for AIX***

There is also an available extension for AIX that covers a restart function, which is shown in Table 1-9.

*Table 1-9 Ansible plug-in for AIX*

<b>Ansible plug-in</b>	<b>Minimum Ansible version</b>	<b>Description</b>
reboot	2.9	Restarts your LPAR.

## **1.5.5 Ansible for IBM i**

Ansible for IBM i bridges existing gaps by providing a versatile solution that aligns with modern requirements. It equips businesses with tools to efficiently automate tasks and manage cloud workload migration. As organizations seek to extract optimum value from their IBM i environments, Ansible offers a practical approach, fostering innovation and adaptability in a dynamic digital landscape.

### **Why Ansible for IBM i**

IBM i has been actively engaged in open-source initiatives since 2008, establishing its presence in domains such as Internet of Things (IoT) and AI. This trajectory is propelled by several influential factors. Open source acts as a channel for IBM i to explore domains such as IoT and AI, with demand surging as a new generation of engineers joins companies that partake in the IBM i ecosystem. These adept developers are positioned to redefine business applications for the IBM i platform.

The acquisition of Red Hat by IBM sparked an interest in Ansible as a tool for automating IBM i processes and tasks. As the adoption of Ansible gains momentum, organizations are seeking IT professionals with expertise in both open-source practices and Ansible capabilities. This intersection creates opportunities for the evolution of the IBM i platform, unlocking avenues for innovation and adaptability.

### **Ansible for IBM i release history**

Ansible for IBM i is on a journey of continuous enhancement and evolution, aligning its capabilities with the ever-evolving landscape of IT requirements and challenges. This roadmap outlines the progressive features and functions that were introduced in different phases, showcasing how Ansible is becoming a part of the IBM i ecosystem.

#### ***June 2020***

In this initial phase, Ansible for IBM i set its foundation by focusing on core operational aspects:

- ▶ Program Temporary Patch (PTF) and Licensed Program Product (LPP) management: Ansible managed PTFs and LPPs efficiently through automation.
- ▶ Open-source package management: The roadmap introduced support for managing open-source packages, which streamlined installation and updates.
- ▶ Object management: Automation was extended to manage objects on the IBM i platform, which promoted a more streamlined and consistent approach.
- ▶ Portable Application Solutions Environment (PASE) support: Ansible embraced the PASE, which enabled more versatile scripting and automation.
- ▶ Work management runtime: Ansible started facilitating work management runtime operations, which enhanced control over system resources.

- ▶ Device management: Automation was expanded to include device management, which helped ensure smoother handling of hardware resources.
- ▶ Independent auxiliary storage pool (IASP) support: The introduction of support for IASPs contributed to enhanced data storage capabilities.

### ***September 2020***

Building on the foundational elements, Ansible for IBM i continued to evolve with a focus on broader capabilities:

- ▶ Advanced patch management: Ansible extended its patch management capabilities, which enabled more advanced and targeted fixes.
- ▶ Basic network configuration: Automation now covered basic network configuration tasks, which helped ensure that network settings were managed effectively.
- ▶ Work management: Work management capabilities were further refined, which enhanced resource allocation and workload management.
- ▶ Security management: Ansible incorporated security management features, which contributed to robust security practices across the platform.
- ▶ Message handling: Automation was introduced to handle message handling tasks, which helped ensure timely and efficient communication.

### ***2021***

As Ansible for IBM i matured, it began addressing more strategic aspects:

- ▶ Solution and product configuration: The focus shifted toward solution and product configuration, which helped enable more comprehensive setups and customization.
- ▶ SQL services bundles: Ansible introduced bundles for SQL services, which streamlined database-related operations.
- ▶ System health bundles: The roadmap incorporated system health bundles, which contributed to proactive system monitoring and maintenance.

### ***2022***

As the landscape continued to evolve, Ansible for IBM i looked ahead:

- ▶ Manage in hybrid cloud: Ansible aimed to offer seamless management capabilities in hybrid cloud environments, which helped ensure consistency across on-premises and cloud deployments.
- ▶ Application management in the cloud: The focus expanded to include application management in cloud environments, which helped enable efficient deployment and maintenance.
- ▶ Enhance existing functions: Ansible continued to enhance its existing functions, which refined automation processes and expanded its capabilities.

### ***Upcoming releases***

Although specific details about upcoming releases are not provided, it is indicated that more use cases and functions will be integrated to further enrich Ansible offerings for IBM i users.

### **Ansible community for IBM i**

The Ansible community for IBM i is a vibrant and active ecosystem that provides a wealth of resources to enhance automation and management capabilities for IBM i. This community-driven effort encompasses a diverse range of modules, roles, and documentation to facilitate integration of Ansible with IBM i environments.

Here is some of the Ansible content for IBM i:

- ▶ 50+ modules available: The Ansible community for IBM i offers a rich repository of over 50 modules, each designed to address specific tasks and functions.
- ▶ 10+ reusable roles: Roles, which are reusable playbooks, play a pivotal role in the Ansible ecosystem. The IBM i community has contributed over 10 roles that encapsulate best practices and standard procedures for commonly performed tasks.
- ▶ Rich resources: The Ansible content for IBM i is supported by a comprehensive set of resources, which helps ensure that users have access to information and guidance for effective implementation:
  - Galaxy: The [Ansible Galaxy URL](#) serves as a centralized repository for IBM i specific content. Here, users can discover, explore, and access many of modules and roles that are tailored for the IBM i platform.
  - GitHub Repository: The [GitHub repository](#) hosts a collection of open-source examples, which showcase practical applications of Ansible automation in IBM i environments.
  - Documentation: The [dedicated documentation portal](#) offers detailed insights into the usage, configuration, and best practices of Ansible for IBM i. This resource guides users through the process of integrating Ansible into their IBM i workflows.
  - Automation Hub: The [Automation Hub](#) that is provided by Red Hat serves as a platform for discovering and sharing Ansible automation content. It offers expertise for users to explore, deploy, and manage Ansible content that is specific to IBM i.

### ***Red Hat Ansible automation on IBM i***

Red Hat Ansible Automation on IBM i represents a powerful synergy between two industry leaders that helps organizations to efficiently manage IBM i workloads within the broader scope of enterprise automation. Red Hat Ansible, a renowned automation platform, offers certified content for IBM Power by providing a robust solution for orchestrating and automating tasks on the IBM i platform. This certified content, which is available through a subscription model, aligns IBM i workloads with the Ansible Automation Platform ecosystem.

Certified integration between Ansible and IBM Power consists of the following components:

- ▶ Unified automation strategy: The integration of Red Hat Ansible with IBM Power offers a certified pathway for holistic automation across a diverse infrastructure landscape. This integration encompasses multiple platforms, including AIX, IBM i, and Linux on Power, which helps enable enterprises to unify their automation strategies under the Ansible Automation Platform.
- ▶ Certified content repository: The Ansible Automation Platform provides a certified repository of content that is tailored for IBM Power. This repository equips organizations with ready-to-use automation playbooks, modules, and roles that efficiently manage and orchestrate IBM i workloads.
- ▶ Solution benefits:
  - Consistency: The certified integration helps ensure uniform automation practices across heterogeneous environments. With Red Hat Ansible Automation, organizations can establish consistent automation workflows that span IBM Power, AIX, and Linux on Power.
  - Transparency: The unified approach that is offered by Red Hat Ansible fosters transparency in automation operations. Organizations gain a comprehensive view of their IBM i workloads alongside other systems, which promotes a clearer understanding of automation tasks and results.

- Skills enhancement: By embracing Red Hat Ansible Automation for IBM i, IT teams can enhance their skill sets and proficiency in modern automation practices. The platform offers a standardized and adaptable automation framework that empowers teams to efficiently manage complex workloads.

**Note:** The integration of Red Hat Ansible Automation with IBM Power reflects a commitment to the automation processes, which improves operational agility and helps enable enterprises to confidently manage their IBM i workloads. This collaboration provides a powerful tool set for organizations that seek to navigate the complexities of enterprise automation with consistency, transparency, and advanced skills development. For more information, see the following resources:

- ▶ [Red Hat Ansible Automation Platform Ecosystem](#)
- ▶ [Ansible Automation Platform Certified and Validated Content](#)

### **IBM i specific Ansible modules**

Within Ansible 2.9 and later, a dedicated set of modules that is tailored specifically for IBM i systems is available, as shown in Table 1-10. Although these modules constitute a smaller subset compared to the complete range that is present in Ansible, they offer targeted functions to cater to the unique requirements of IBM i environments.

*Table 1-10 Specific Ansible modules for IBM i*

<b>Module</b>	<b>Minimum Ansible version</b>	<b>Description</b>
ibmi_at	2.9	Schedules a batch job on a remote IBM i node.
ibmi_cl_command	2.9	Runs a CLI command.
ibmi_copy	2.9	Copies a save file from a local node to a remote IBM i node.
ibmi_display_subsystem	2.9	Displays all active subsystems or active jobs in a subsystem.
ibmi_end_subsystem	2.9	Ends a subsystem.
ibmi_start_subsystem	2.9	Starts a subsystem.
ibmi_lib_restore	2.9	Restores one library on a remote IBM i node.
ibmi_lib_save	2.9	Saves one library on a remote IBM i node.
ibmi_reboot	2.9	Restarts the IBM i machine.
ibmi_sql_execute	2.9	Runs an SQL non-DQL (Data Query Language) statement.
ibmi_sql_query	2.9	Runs an SQL DQL (Data Query Language) statement.
ibmi_patch	2.9	Loads from a save file, and applies, removes, or queries PTFs.
ibmi_patch_imgclg	2.9	Installs fixes from a virtual image.
ibmi_object_find	2.9	Finds a specific IBM i object.

Module	Minimum Ansible version	Description
ibmi_submit_job	2.9	Submits an IBM i job.
ibmi_iasp	2.9	Controls an IASP on a target IBM i node.
ibmi_tcp_interface	2.9	Manages the IBM i TCP interface. You can add, remove, start, end, or query a TCP interface.
ibmi_tcp_server_service	2.9	Manages a TCP server on a remote IBM i node.

### **Shared core modules with IBM i support**

These modules are named “common modules”, and they are compatible with many OSs. These modules are harnessed by IBM i too. Core modules extend their support to IBM i through PASE.

Table 1-11 shows the shared core modules with IBM i compatibility.

*Table 1-11 Shared core common modules that are supported on IBM i*

Common modules	Minimum Ansible version	Description
assemble	2.9	Assembles content from different sources into a single file or variable.
authorize_key	2.9	Adds or removes SSH authorized keys for specified users.
blockinfile	2.9	Inserts or updates a block of text that is surrounded by customizable markers.
command	2.9	Runs shell commands on target hosts.
copy	2.9	Copies files to remote locations.
fetch	2.9	Fetches files from remote locations.
file	2.9	Manages files and file properties on remote hosts.
find	2.9	Searches for files in a directory hierarchy.
git	2.9	Manages Git repositories on remote hosts.
lineinfile	2.9	Helps ensures that a particular line is in a file, or replaces an existing line.
pause	2.9	Pauses a playbook for a specified amount of time.



Common modules	Minimum Ansible version	Description
ping	2.9	A basic connectivity test to target hosts.
pip	2.9	Manages Python packages by using pip.
script	2.9	Runs a local script on the remote hosts.
setup	2.9	Gathers system information from target hosts.
shell	2.9	Runs shell commands on target hosts.
stats	2.9	Gathers facts about remote hosts.
synchronize	2.9	Synchronizes files and directories to remote hosts.
wait_for_connection	2.9	Waits for a host to become reachable.

### ***Specific Ansible roles for IBM i***

Ansible Role encompasses a collection of tasks that is designed to configure an IBM i host for various common tasks. These tasks include activities such as applying PTFs and other configuration procedures on the IBM i platform. Roles are articulated by using YAML files within a structured directory layout. This layout typically comprises sections such as defaults, vars, tasks, files, templates, and more.

Table 1-12 shows a selection of Ansible roles that are tailored for IBM i.

*Table 1-12 Specific Ansible roles for IBM i*

Ansible role	Minimum Ansible version	Description
apply_all_loaded_ptfs	2.9	Applies all loaded PTFs on IBM i hosts.
apply_ptf	2.9	Applies specific PTFs on IBM i hosts.
change_server_state_via_powervc	2.9	Changes the server state by using PowerVC on IBM i hosts.
configure_passwordless_ssh_login	2.9	Configures a passwordless SSH login on IBM i hosts.
deploy_vm_via_powervc	2.9	Deploys IBM i VMs in PowerVC.
display_network_info_via_powervc	2.9	Displays network information through PowerVC on IBM i hosts.
display_vm_info_via_powervc	2.9	Displays VM information through PowerVC on IBM i hosts.

Ansible role	Minimum Ansible version	Description
download_individual_ptfs	2.9	Downloads individual PTFs on IBM i.
present_ip_interface	2.9	Presents an IP interface configuration on IBM i hosts.

**Note:** To explore more Ansible roles that are tailored for IBM i, see this [comprehensive collection](#). This repository hosts many roles to facilitate IBM i specific tasks, and provides a valuable resource for enhancing your automation capabilities on the platform.

### ***Ansible extensions for IBM i***

Ansible extensions for IBM i are composed of code components that complement the core functions of Ansible, which enrich its capabilities. These extensions, which often are referred to as plug-ins, serve as dynamic tools to enhance flexibility and expand the feature set of Ansible.

**Note:** Finding these plug-ins is straightforward. For more information, see the following resources:

- ▶ Ansible Galaxy: [IBM Power i Plug-ins](#)
- ▶ GitHub repository: [Ansible IBM i Plug-ins](#)

Table 1-13 shows a selection of Ansible plug-ins that are designed for IBM i. These plug-ins cover various functions, such as copying files, interacting with IBM Db2® on IBM i, utility functions, fetching data, and restarting operations.

*Table 1-13 Ansible plug-ins for IBM i*

Ansible plug-in	Minimum Ansible version	Description
ibmi_copy	2.9	Copies files and directories on the IBM i system.
ibmi_db2i_tools	2.9	Provides tools for managing Db2 on IBM i.
ibmi_ibmi_module	2.9	Offers various IBM i specific modules for Ansible tasks.
ibmi_ibmi_util	2.9	Includes utility functions for IBM i tasks.
ibmi_fetch	2.9	Fetches files and URLs on the IBM i system.
ibmi_reboot	2.9	Initiates system restarts on the IBM i platform.

### ***Ansible playbooks for IBM i***

An Ansible playbook for IBM i serves as a structured set of automation tasks to run with minimal to no human intervention. These playbooks, which are crafted in YAML format, consist of mappings and sequences, and they incorporate Ansible modules to run specific actions. A prime feature of Ansible playbooks is their ability to automate intricate workflows on the IBM i platform.

Table 1-14 illustrates a selection of Ansible playbooks that are tailored for IBM i.

Table 1-14 Ansible playbooks for IBM i

Ansible playbooks	Minimum Ansible version	Description
enable-ansible-for-i.yml	2.9	Configures prerequisites on IBM i endpoints for using Ansible for IBM i collections.
enable_offline_ibmi.yml	2.9	Enables Ansible automation for IBM i endpoints without requiring internet access.
ibmi-sql-sample.yml	2.9	Demonstrates the usage of Ansible for IBM i to run SQL statements on IBM i systems.
ibmi-sysval-sample.yml	2.9	Illustrates Ansible for IBM i to query system values on IBM i platforms.
ssh-addkey.yml	2.9	Facilitates the addition of SSH keys on IBM i systems, helping enable secure communication for Ansible operations.

**Note:** To explore practical examples, see [GitHub](#).

### ***Ansible inventory on IBM i***

An Ansible inventory serves as a configuration file that defines individual hosts and groups of hosts within your environment. You can manage multiple systems within your infrastructure simultaneously. For example, you can create inventory groups that reflect different parts of your infrastructure to target specific hosts or sets of hosts.

In the context of IBM i, your inventory file can define various groups such as “IBM Power Virtualization Center” and “IBM i systems”, which provides a clear structure for managing and orchestrating tasks across different systems. Each group is associated with specific attributes, which include connection details and authentication credentials.

Example 1-17 shows an inventory setup where you have two groups: “powervc\_servers” and “IBM i”. The “powervc\_servers” group includes details about the Power Virtualization Center servers, which specify the SSH host, username, password, and Python interpreter. The “IBM i” group outlines the connection information for your IBM i system.

Example 1-17 Sample Ansible inventory configuration for IBM i and PowerVC

---

```
[powervc_servers]
powervc ansible_ssh_host=your_powervc_ip ansible_ssh_user=your_user
ansible_ssh_pass=your_password
ansible_python_interpreter="python3"

[ibmi]
source ansible_ssh_host=your_source_ibmi_ip
ansible_ssh_user=your_user ansible_ssh_pass=your_password
```

---

By using this approach, you can efficiently manage and automate tasks across a diverse set of systems within your infrastructure. This approach helps ensure that Ansible can interact with the specified hosts for configuration management and orchestration.

## 1.5.6 Ansible for IBM Power Hardware Management Console

The IBM Power HMC is rapidly evolving, with major advances being made across Power servers. Automation is of paramount importance to save time and increase efficiency. The `power-hmc` collection is part of IBM continuous efforts to adapt HMC to the ever-evolving automation trends in the IT infrastructure administration landscape.

Power HMC Ansible content helps administrators include Power HMC as part of their automation strategies through the Ansible ecosystem. By using Power HMC, Ansible content in IT automation helps maintain a consistent and convenient management interface for multiple Power HMCs and Power servers.

Power HMC Ansible content modules can be leveraged to do Power HMC patch management, LPAR management, Power server management, password policy configurations and HMC-based Power server dynamic inventory building.

### ***HMC specific Ansible modules***

Table 1-15 provides a summary of the modules that are supported at the time of writing.

*Table 1-15 Specific Ansible modules for the IBM Power Hardware Management Console*

<b>Module</b>	<b>Minimum Ansible version</b>	<b>Description</b>
<code>hmc_command</code>	2.9	Use this module to run HMC commands.
<code>hmc_pwdpolicy</code>	2.9	The password policy module manages the Power HMC password policy.
<code>hmc_update_upgrade</code>	2.9	Use the HMC patch management module to update or upgrade the HMC.
<code>hmc_user</code>	2.9	Manage the users on the HMC.
<code>power_system</code>	2.9	Use the Power management module to power cycle the system, modify configurations, and modify resources.
<code>powervm_dlpar</code>	2.9	Use this module to dynamically configure the Processor, Memory, and Storage settings.
<code>powervm_inventory</code>	2.9	A dynamic inventory plug-in that dynamically builds the inventory of Power servers and partitions that are connected to the HMC.
<code>powervm_lpar_instance</code>	2.9	The LPAR management module helps with the creation, deletion, activation, and shutdown of LPARs.
<code>powervm_lpar_migration</code>	2.9	Use the LPAR migration module to validate and migrate LPARs.
<code>vios</code>	2.9	The Virtual IO Server module helps to create and install VIOS.

## ***Ansible extensions for the HMC***

There are also extensions that are available for the HMC, as shown in Table 1-16.

*Table 1-16 Ansible plug-ins for the HMC*

<b>Ansible plug-in</b>	<b>Minimum Ansible version</b>	<b>Description</b>
hmc_resource	2.9	Removes try from the <code>list_all_managed_system_details</code> method.
hmc_exceptions	2.9	Changes the custom exception name.
hmc_rest_client	2.9	Adds unit tests and document correction.
hmc_cli_client	2.9	Patches the password special character issue.
hmc_command_stack	2.9	Implements a patch for <code>show ldap details</code> in the <code>configure ldap</code> action.
powervm_inventory	2.9	HMC-based inventory source for Power servers.

The `ansible-power-hmc` collection requires that you are running HMC 10, HMC 9.1 or later, or HMC 8.8.7.0 or later. It supports Ansible 2.9 or later and requires Python 3.

For more information about the collection, see [Ansible Collection for Power HMC](#).

## **1.5.7 Ansible for Power Virtual I/O server**

IBM PowerVM VIOS is a special IBM AIX based appliance for IBM Power that helps virtualize storage and network adapters for VMs that run on the managed system. The VIOS should be installed in pairs to provide a HA enterprise system. In addition, you might have business requirements for the separation of applications that requires more VIOS pairs for security. Therefore, have at least two VIOS partitions for each physical server that you manage.

This approach is perfect for Ansible management because you must help ensure that all your VIOS LPARs are maintained and updated as required as new updates come out. Also, if you are building a new bare metal environment, it is helpful to help ensure that the VIOS image that is used in the new server is the level that you want. Starting with VIOS 4.1, VIOS is already enabled for Ansible, which makes your Ansible setup simpler.

The `power-vios` collection provides the functions that you need to manage your VIOS environment, such as install a VIOS, manage the security parameters on your VIOS, back up and restore VIOS images, and upgrade the VIOS code.

### ***VIOS specific Ansible modules***

Table 1-17 provides a summary of modules that are supported at the time of writing.

*Table 1-17 Specific Ansible modules for VIOS*

<b>Module</b>	<b>Minimum Ansible version</b>	<b>Description</b>
mapping_facts	2.9	Returns the mapping between physical, logical, and virtual devices as facts.
updatevios	2.9	Updates the VIOS to the latest maintenance level.
viosbr	2.9	Backs up and restores the configuration of the VIOS.

Module	Minimum Ansible version	Description
backupvios	2.9	Creates an installable image of the root volume group.
viosupgrade	2.9	Upgrades the VIOS.
alt_root_vg	2.9	Creates and cleans up an alternative rootvg disk on a VIOS.
viosecure	2.9	Configures security hardening rules and a firewall.

### ***Ansible extensions for the VIOS***

There is also an extension that is available for the VIOS, as shown in Table 1-18.

*Table 1-18 Ansible plug-ins for the VIOS*

Ansible plug-in	Minimum Ansible version	Description
viosupgrade	2.9	Updates viosupgrade.py.

## **1.5.8 Ansible for IBM Power Systems Virtual Server**

IBM Power Systems Virtual Server on IBM Cloud offers a rapid and efficient means to create and deploy VMs across various OSs, including AIX, IBM i, and Linux that is tailored for Power. This solution stands out for its robust security measures and the ability to scale compute capacity as needed.

### ***Enterprise Infrastructure as a Service offering***

This service uses Power resources that are distributed globally, which helps ensure low-latency connectivity to IBM Cloud infrastructure. It features dedicated components, which include a distinct network and direct attached storage, which enhances reliability and performance.

### ***Collaborative Cloud offering***

At its core, this offering represents an Infrastructure as a Service (IaaS) approach, which reflects the evolving IT landscape where cloud capabilities are essential for resource consumption. IBM recognizes that its extensive client base relies on AIX and IBM i, which this solution a vital addition.

### **Architectural overview**

Within the IBM Cloud environment, the IBM Power Systems Virtual Server is as a distinct entity. Picture it as a specialized colocation (COLO) site within an IBM SoftLayer® or cloud data center. Here, you find a segregated enclosure that houses all the Power equipment. An advantage of IBM Power Systems Virtual Servers is the consistency of their architecture with on-premises Power servers. If you are configuring Power servers on-premises today, it is a similar process to set up an IBM Power Systems Virtual Server. To illustrate the architecture, see Figure 1-12 on page 55.

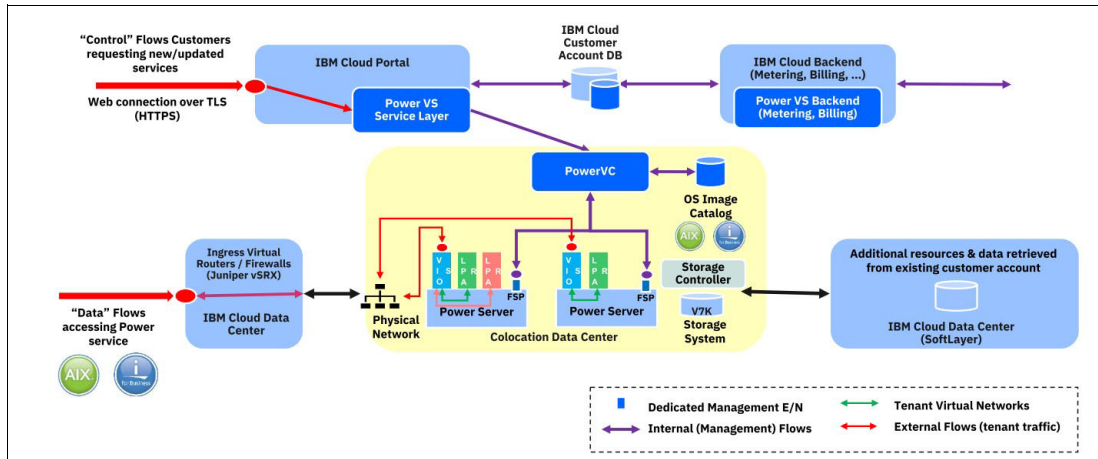


Figure 1-12 Architecture of the service

For example, redundant VIOSs and NPIV-attached storage, as seen in on-premises setups, are mirrored in Power Systems Virtual Server environments. Redundant storage area network (SAN) fabric, HMC, and PowerVC configurations are all consistent between on-premises and cloud deployments.

### Infrastructure as a Service

Moving Power resources to a cloud data center is not a simple relocation, which is why you need a COLO site. The COLO site serves as the foundation for IaaS. In this context, IaaS encompasses everything beneath the VM layer, which includes the PowerVM hypervisor, firmware, VIOS, HMC, PowerVC, network switches, and SAN switches, all of which are part of the IaaS.

So, how do you manage this environment if you cannot directly access PowerVC or the SAN switches? *You interface with a service layer that implements the open service broker framework.* This framework is a standard across various cloud portals, such as Google Cloud Platform and Azure. Essentially, it is a means of managing services in the cloud.

This service layer offers multiple interfaces, including a CLI and a REST API. Although the interface might change, you retain the same core capabilities that you are accustomed to on-premises. For example, if you have scripts that interact with the HMC today, you must adapt them to the IBM Cloud CLI. However, the functions that you rely on remain available.

### Ansible integration with IBM Power Systems Virtual Server

Within the Power Systems Virtual Server architecture, PowerVC plays a central role. It enables integration with various DevOps tools like Terraform, Ansible, and Puppet, and offers an API for enhanced automation capabilities.

With Ansible, you can provision AIX and IBM i instances within IBM Power Systems Virtual Server.

### IBM Power Systems Virtual Server in IBM Cloud

In this example, we demonstrate the creation of a Power Systems Virtual Server running AIX or IBM i. This server is configured to allow incoming SSH connections through a publicly accessible IP address, which is authenticated by using the provided SSH key.

## ***Power Systems Virtual Server resources***

The following infrastructure resources are established by using Ansible modules:

- ▶ SSH Key (`ibm_pi_key`)
- ▶ Network (`ibm_pi_network`)
- ▶ Virtual Server Instance (VSI) (`ibm_pi_instance`)

### ***Configuration parameters***

Users may set the following parameters:

- ▶ `pi_name`: The name that is assigned to the VSI.
- ▶ `sys_type`: The type of system on which to create the VM (for example, s922, e880, or any).
- ▶ `pi_image`: The name of the VM image (users can retrieve available images).
- ▶ `proc_type`: The type of processor mode in which the VM runs (shared or dedicated).
- ▶ `processors`: The number of vCPUs to assign to the VM (as visible within the guest OS).
- ▶ `memory`: The amount of memory (in GB) to assign to the VM.
- ▶ `pi_cloud_instance_id`: The `cloud_instance_id` for this account.
- ▶ `ssh_public_key`: The value of the SSH public key that is authorized for SSH access.

### ***Running the playbook***

Before proceeding, help ensure that you set your API Key and Region by completing the following steps:

1. Obtain an IBM Cloud API key.
2. Export your API key to the `IC_API_KEY` environment variable by running the following command:

```
export IC_API_KEY=<YOUR_API_KEY_HERE>
```

**Note:** Although modules also support the `ibmcloud_api_key` parameter, use environment variables when encrypting your API key value.

3. Export your IBM Cloud region to the `IC_REGION` environment variable by running the following command:

```
export IC_REGION=<REGION_NAME_HERE>
```

**Note:** Modules also support the `region` parameter.

4. Export your IBM Cloud zone to the `IC_ZONE` environment variable by running the following command:

```
export IC_ZONE=<ZONE_NAME_HERE>
```

**Note:** This environment variable is used for multi-zone supported power instances.

With the environment variables configured, create all the resources and test the public SSH connections to the VM, and then run the `create` playbook.

## **1.5.9 Ansible for applications**

Ansible can simplify the installation and operation of your infrastructure components, including creating LPARs running AIX, IBM i, or Linux on Power. However, Ansible can also provide a reduction in the time and effort that is required to manage many complex application environments.



Using Ansible for automation in these application environments can reduce the amount of time that your support staff spends on basic tasks like installing or modifying your application environment. You can have a build a consistent and secure environment for your application instances while allowing your support staff to concentrate on tasks that are more important and that can drive new business opportunities for your company. There are many application environments that can be managed by Ansible. We describe how Ansible can make your team more productive when managing two of the most common applications that run on IBM Power servers.

## **Ansible for Oracle**

Oracle Database has been a leading enterprise database management system for over 3 decades. Despite the emergence of new technologies and competitors, Oracle Database remains a popular choice for many organizations.

Oracle Database manages and processes large amounts of data quickly and efficiently. As a result, Oracle Database is widely used across different industries, from finance to healthcare, and is trusted by organizations of all sizes to store and manage critical business data. Oracle Database established itself as a reliable and versatile database management system that can meet the complex data needs of modern enterprises. Its performance capabilities, scalability, security features, and integration with cloud technologies make it a top choice for organizations.

One of the most common platforms for running Oracle databases and application is on IBM Power running the AIX OS. With automation becoming the norm in IT operations, installation and administration of Oracle Database on AIX is not an exception. There are Ansible collections for installation operations both on a single-node machine and on Oracle Real Application Cluster (RAC) in addition to a collection for automating database administrator (DBA) operations.

### ***Advantages of using Ansible for Oracle***

The Ansible collections for Oracle on AIX provide the following benefits:

- ▶ **Automated database installation**

Even when creating a single instance database, setting up Oracle Database on AIX involves multiple manual processes, including defining the LPAR, installing AIX, configuring the file system (either JFS2 or Automatic Storage Management (ASM)), configuring the network connections, and setting up system configuration values that are appropriate for running your database environment.

Setting up an Oracle RAC on AIX involves setting up an AIX environment on the hosts that meet RAC's specific requirements from kernel tunables, network attributes, shared disk attributes, passwordless to user equivalent SSH connections and other items. The manual process to accomplish these tasks is tedious and error-prone. During the Grid and Database installation, the GUI frequently prompts for input, which occupies the user for a long time.

You can save time with infrastructure automation. The whole installation can take 2 days for seasoned users. With the help of the Ansible Oracle RAC ASM collection, it takes about 5 hours to complete a 4-node RAC installation. It is automated and can consistently re-create an Oracle RAC for other projects. The value of this collection helps your organization to improve productivity for installation tasks.

► Automated database management

After your databases are installed and operational, there are many tasks that must be done to manage the environments. As the number of database instances grows, the amount of time that is required for day-to-day operations also grows. The Ansible Oracle DBA package can automate the day-to-day tasks that are managed by your DBAs to increase their productivity and focus on more business-critical issues.

With the ODBA collection, you can add or drop databases, manage users, manage space (ASM and ASM Cluster File System (ACFS) functions), manage patches, and otherwise manage your Oracle environment. In addition, the ODBA collection can automate Oracle upgrades.

### Ansible for SAP

Ansible is an open-source automation tool that is used to simplify the management and deployment of IT infrastructures. It is especially useful in managing complex systems like SAP. In today's digital world, if an organization relies on SAP HANA and SAP S/4HANA for its business-critical operations, downtime can result in revenue loss, performance and service degradation, increased security exposure, and poor user experiences. Your ROI is affected and your SAP teams are distracted from more strategic, high-priority projects.

Figure 1-13 illustrates the breadth of function that can be provided by Ansible for automating your SAP landscape.

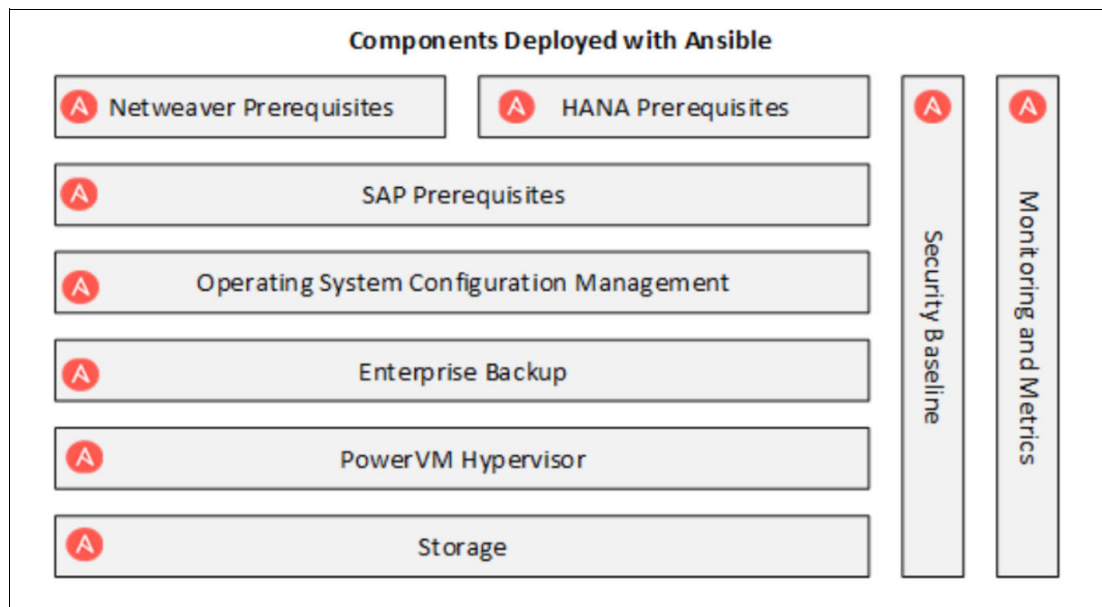


Figure 1-13 Components that can be automated by using Ansible<sup>16</sup>

Ansible Automation Platform eliminates these common obstacles with an intuitive interface and trusted content that is custom built for SAP migrations. Ansible can also be integrated with SAP to streamline operations, improve efficiency, and reduce manual tasks. With Ansible Automation Platform, manual tasks that used to take days can be done in hours or even minutes. By consolidating on a single, unified platform, your teams can more easily share automation content and workflows, and scale as your organization evolves and uncovers new automation use cases.

<sup>16</sup> Source: <https://www.adventone.com/sap-hana-at-the-speed-of-ansible/>

SAP stands for Systems, Applications, and Products in Data Processing. It is a software suite that covers various business processes, such as finance, logistics, human resources, and more. SAP is used by organizations of all sizes and industries to manage their operations effectively.

SAP installations can be complex and require skilled administrators to manage and maintain them, which usually involves performing various tasks such as system configuration, installation of patches, managing user accounts, monitoring system performance, and more.

### ***Advantages of using Ansible for SAP***

The following list counts some of the advantages of using Ansible to manage your SAP environment:

▶ **Rapid deployment and configuration**

Deploying and configuring SAP systems, such as SAP HANA and S/4HANA, can be complex and time-consuming. Ansible simplifies this process by automating the deployment of SAP software, along with the necessary configurations. With Ansible playbooks, you can define the system settings, network configurations, and more, which helps ensure consistency across all your SAP systems. Furthermore, Ansible has reusability, so you can replicate configurations across different environments.

▶ **Continuous software delivery**

Ansible streamlines the software deployment process, which helps enable CD of updates, patches, and enhancements. With Ansible playbooks, you can automate the entire software installation, patching, and upgrade processes. Ansible can perform tasks in parallel to save time by running these operations on multiple systems simultaneously. This approach helps ensure that your SAP systems are up to date with minimal disruption and maximum efficiency.

▶ **Intelligent monitoring and maintenance**

Monitoring your SAP landscape is crucial for identifying potential issues and preventing system downtime. Ansible integrates seamlessly with monitoring tools, so you can automate monitoring processes and trigger alerts or actions based on predefined thresholds. By proactively monitoring performance metrics, system errors, and other vital indicators, you can help ensure the availability and reliability of your SAP environment.

▶ **Enhanced high availability and disaster recovery (HADR)**

Ensuring that capabilities for SAP systems is paramount for minimizing business interruptions. With Ansible, you can automate the configuration and management of HADR environments. Whether Ansible is setting up system replication between primary and secondary nodes or automating failover and failback processes, it streamlines these tasks and reduces the risk of human errors.

▶ **Scalability and flexibility**

With Ansible, you can scale your SAP infrastructure to handle growing business demands efficiently. With Ansible playbooks, you can define rules and conditions for scaling up or down system resources, such as adding or removing compute nodes or adjusting memory allocations. This flexibility optimizes resource utilization and accommodates changing workloads effortlessly.

Managing SAP systems can be complex and time-consuming, but with the power of Ansible, organizations can revolutionize their SAP operations. By automating tasks such as system configuration, software deployment, monitoring, and scaling, Ansible simplifies SAP management, enhances efficiency, and drives agility in your SAP landscape. With the Ansible Automation Platform, you can streamline your SAP operations, reduce manual effort, help ensure consistency, and improve the overall productivity and reliability of your SAP environment.



## Ansible architecture and design

Understanding how to set up Ansible to manage automation in your environment is important to the ultimate success of your automation journey. Set up the Ansible environment to support your staff and meet your business needs.

This chapter describes the different components of Ansible and presents some hints and tips for you to consider as you build your Ansible environment.

The following topics are described in this chapter:

- ▶ Ansible architecture and components
- ▶ Understanding the Ansible declarative language
- ▶ Understanding an Ansible inventory
- ▶ Ansible tasks, playbooks, and modules
- ▶ Ansible roles and collections
- ▶ Best practices for playbook and role design
- ▶ Creating versions and documenting playbooks and roles
- ▶ Testing and validating playbooks and roles

## 2.1 Ansible architecture and components

In early releases, Ansible was installed as a package that included core components and plug-ins that you used to control automation on different platforms. This approach was too complex to maintain, which introduced more difficulties and delays in updating the Ansible package. Now, Ansible is released in multiple packages that can be installed independently.

Figure 2-1 shows some of the Ansible components.

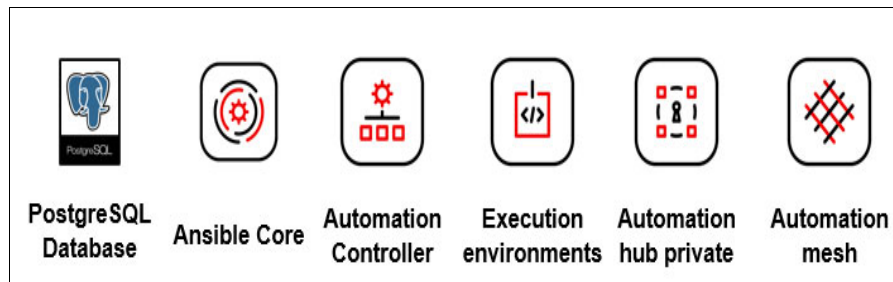


Figure 2-1 List of Ansible automation components

One option for building your environment is to use the automation base package Ansible Core (previously called Ansible Engine or Ansible base) and use Ansible Automation Controller as the CLI.

Alternatively, the automation environment can be built with Red Hat Ansible Automation Platform together with Ansible Core. Automation Platform has a GUI and also extends Ansible functions with more management capabilities. For more information, see 1.3.3, “Ansible Automation Platform” on page 10.

Consider several things before deciding how to set up your automation environment. These considerations help you to choose the right Ansible products as you define and build the proper architecture that meets your business requirements.

- ▶ **Computing resource availability and cost**  
Understand the number of systems that will be managed and the required availability of your Ansible management infrastructure. This item determines the number of Ansible systems that are required to manage your environment, including the required resources (CPU, memory, and disk).
- ▶ **Administrative environment**  
Do you need a CLI or a GUI?
- ▶ **Security and compliance**  
Understand your security and compliance requirements for user authentication and access control. Do you need to separate automation into multiple management domains to comply with security and compliance guidelines?
- ▶ **High availability (HA) and scalability**  
Do you need HA in your automaton? If you are monitoring and managing critical business functions, the answer is probably yes. Design and build the automation so that it scales to manage more environments or handles growth in the number of machines and tasks running.

- ▶ Integration and compatibilities

Do you need to integrate with other solutions and services? Is your solution compatible with the hosts and systems that are targeted for automation?

- ▶ Complexity of Day 2 operations

Understand how complex it is to manage the automation environment and operational activities. Especially consider automation resources like the following ones:

- Credentials to access target systems.
- Skilled resources to develop and manage playbooks.
- Take time to define inventories, hosts, and hosts groups to simplify the automation environment.
- Resources to manage playbook execution and access control.

When you have considered all of these factors, plan and design the automation environment based on your requirements in these areas. For more information about designing your Ansible environment, including reference architectures, see 3.1, “Designing your Ansible environment” on page 104.

## 2.1.1 Controller and client functions

From an administrative view, there are two primary functions of all the systems in the automation environment. A system is either a controller, or a system is a client, that is, a system that is managed.

### Controller functions

A *controller* is the machine or set of machines that run the Ansible tools (`ansible-playbook`, Ansible, `ansible-vault`, and others) and automation tasks. Depending on your solution, the controller uses the Ansible CLI or a GUI. Using a GUI often simplifies the management of your client inventory, job templates, and workflow templates, and simplifies how you start jobs, schedule workflows, and track and report changes. For more information about the requirements for an Ansible Controller, see 3.2, “Choosing the Ansible Controller node” on page 121.

### Client functions

A *client* is a machine or set of machines that Ansible manages. They are also referred to as 'hosts' or managed nodes, and are servers, network appliances, or other managed devices. Ansible does not need to be installed on the managed nodes because the Ansible control nodes are used to run the automation tasks. For more information about the prerequisites for a managed node, see 3.4, “Preparing your systems to be Ansible clients” on page 149.

### Controller to client connectivity

The Ansible Controller is responsible for automating task execution on the managed nodes or target devices (servers, network appliances, or any computer), Ansible works by connecting the controller to the managed nodes and pushing out small programs that are called Ansible modules to them. These programs are written to define the needed state of the client and then prompt the client to make the required changes to help ensure that it meets the needed state.

The communication between Ansible Controllers to managed nodes or the target devices can be different depending on the target devices. Here are some examples:

- ▶ Linux and UNIX hosts use Secure Shell (SSH) by default.
- ▶ Windows hosts use WinRM over HTTP/HTTPS by default.
- ▶ Network devices use CLI or XML over SSH.
- ▶ Appliance or web-based services use a Representational State Transfer (REST) application programming interface (API) over HTTP/HTTPS.

## 2.2 Understanding the Ansible declarative language

Ansible playbooks, which are used to define automation tasks, are written in YAML. YAML is used to describe the tasks, expected configuration, and data structures in a human-readable format. YAML is a recursive acronym for YAML Ain't Markup Language.

This section explores how Ansible uses YAML to define automation playbooks. We provide some simple examples to help you get started with effective building automation tasks.

### 2.2.1 YAML structure

YAML creates a human-readable format. YAML files consist of key-value pairs, lists, and nested structures. Indentation is important in YAML because indentation is used to define the hierarchy of commands and lists.

When you create a YAML file, consider these basic rules:

- ▶ YAML is case-sensitive.
- ▶ The files should have `.yaml` or `.yml` as the extension.
- ▶ Indentation is critical to define the hierarchy of the YAML code:
  - Do not use tabs for indentation. Use spaces.
  - As a best practice, use two spaces per indent. Many editors can be set to provide spacing when editing YAML files.
- ▶ Playbooks start with `---` to denote the beginning of a document and end with `...` to denote the end of a document.
- ▶ There are important key sequences to define Ansible code:
  - Comments are denoted by a space that is followed by a hashtag (`#`). Any text that follows the space hashtag is ignored.
  - A colon followed by a space (or newline) `:`: `"` is an indicator for a mapping.
  - To have any of these items in a string and not be interpreted by Ansible, use single quotation marks or double quotation marks around the full value string.
- ▶ Extra lines are ignored and can be used to improve the readability of the playbook.



For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key-value pairs, commonly called a “hash” or a “dictionary”. So, you must know how to write lists and dictionaries in YAML:

- ▶ All members of a list are lines that begin at the same indentation level and start with a “- ” (a dash and a space), as shown in Example 2-1.

*Example 2-1* *YAML list definition*

---

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

---

- ▶ A dictionary is represented in a simple <key: > value form (a space must follow the colon), as shown in Example 2-2.

*Example 2-2* *YAML dictionary definition*

---

```
---
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
...
```

---

In the playbook in Example 2-3, the lines `name` and `hosts` are simple key-value pairs. The line `tasks` is a list. A list can contain other objects that are defined with key-value pairs that are organized in a hierarchy.

*Example 2-3* *Example playbook*

---

```
---
- name: Simple Ansible Playbook
  hosts: web_servers
  tasks:
    - name: Ensure that Apache is installed
      apt:
        name: apache2
        state: present

    - name: Start Apache service
      service:
        name: apache2
        state: started
...
```

---

## Variables

You can define variables in Ansible to make your playbooks more dynamic and reusable. Variables can be defined at different levels, including playbook, group, and host variables. In Example 2-4, the vars section defines a playbook-level variable <http\_port>.

The template module is used to copy a configuration template (~apache.conf.j2) to the destination and notify the ~Restart Apache` handler. You might notice j2 in the code example, which refers to the Jinja templating language that is common in Python development. For more information about Jinja templating, see 2.2.2, “Jinja2” on page 67.

*Example 2-4 Playbook with variables*

---

```
---
- name: Ansible Playbook with Variables
  hosts: web_servers
  vars:
    http_port: 80
  tasks:
    - name: Ensure that Apache is installed
      apt:
        name: apache2
        state: present

    - name: Configure Apache
      template:
        src: apache.conf.j2
        dest: /etc/apache2/apache2.conf
      notify: Restart Apache

    - name: Start Apache service
      service:
        name: apache2
        state: started
  handlers:
    - name: Restart Apache
      service:
        name: apache2
        state: restarted
```

---

## Conditionals and loops

Ansible supports conditionals and loops to make your playbooks more flexible. Example 2-5 shows a simple example.

In this playbook, <http\_ports> is a variable that is a list of ports. The loop keyword is used to iterate over the list and generate configuration files for each port.

*Example 2-5 Playbook with a variable and loop*

---

```
---
- name: Ansible Playbook with Conditionals and Loops
  hosts: web_servers
  vars:
    http_ports:
      - 80
      - 8080
  tasks:
```

```

- name: Ensure that Apache is installed
  apt:
    name: apache2
    state: present

- name: Configure Apache
  template:
    src: apache.conf.j2
    dest: "/etc/apache2/apache{{ item }}.conf"
    loop: "{{ http_ports }}"
    notify: Restart Apache

- name: Start Apache service
  service:
    name: apache2
    state: started
handlers:
- name: Restart Apache
  service:
    name: apache2
    state: restarted

```

---

## 2.2.2 Jinja2

Jinja is a template engine for Python. It was developed by Armin Ronacher, the creator of the Flask web framework, and was first released in 2008. Jinja2 is used for generating dynamic content, such as HTML, XML, JSON, YAML, and other text-based formats by incorporating data from various sources into predefined templates.

Jinja2 was created as a successor to the original Jinja template engine. The original Jinja was inspired by the Django template system, but is more flexible and extensible. However, it had some limitations, and Jinja2 was developed to address these shortcomings and provide a more powerful, feature-rich, and robust templating engine.

Jinja2 quickly gained popularity within the Python community due to its simplicity, readability, and performance. It became the default template engine for Flask, a popular web framework, which contributed to its widespread adoption.

Jinja2 offers several features and benefits that make it a valuable tool in various contexts:

- Template Inheritance** Jinja2 supports template inheritance. You can create a base template with common structure and blocks. Child templates can extend the base template and override specific blocks, promoting code reuse and maintainability.
- Variables** You can easily insert variables into templates by using double curly braces `{{ ... }}`, which enables dynamic content generation by substituting placeholders with actual data from variables.
- Control Structures** Jinja2 provides control structures like `if` and `for`, and macros, which you use to create conditional logic, loops, and reusable template fragments.

<b>Filters</b>	With filters, you can modify variables before displaying them in templates. Filters can format dates, convert strings to uppercase, sort lists, and perform various other operations on data.
<b>Extensibility</b>	Jinja2 can be extended with custom filters, tests, and extensions, making it highly adaptable to specific requirements and use cases.

Jinja templating with YAML makes Ansible a powerful automation platform that enhances customization, readability, and re-usability of templates.

For more information, see [Template Designer Documentation](#).

## 2.3 Understanding an Ansible inventory

Ansible is a powerful open-source automation tool that simplifies the management of complex IT systems. Users can define and deploy automation tasks through simple, human-readable YAML scripts. One crucial aspect of working with Ansible is understanding the concept of an inventory.

If you want to manage your servers and applications with Ansible, you must define a list of them. This list of targets is called an *inventory* in Ansible. It can be a simple static inventory that is similar to a list of ingredients that you want to buy in a shop, or it can be a more complex static inventory with groups and more variables.

The inventory can also be a dynamic inventory where you get the list from a third-party provider like IBM Cloud, IBM PowerVC, or IBM Power Hardware Management Console (HMC).

### 2.3.1 Overview of an Ansible inventory

In simple terms, an *inventory* is a file or group of files that lists all the remote hosts (machines) that Ansible manages. It serves as a *source of truth* for Ansible, and can identify target systems and run specific tasks on them. The inventory file can be formatted as plain text, INI, or YAML, and it is named `inventory` or `hosts`.

#### Inventory components

Beyond identifying remote hosts, an inventory can be more than a list of servers. It can also include information about groups, variables, and more. Here are some of those key components:

<b>Hosts</b>	Represent the individual machines or servers that you want to manage with Ansible. Each host typically has a unique name or IP address that is associated with it.
<b>Groups</b>	Organizing hosts into groups makes it simpler to manage and perform operations on specific sets of machines. For example, you can have groups like webservers, database-servers, or even staging and production groups.
<b>Variables</b>	With an inventory, you can define variables at both the host and group level. Use these variables to customize and parameterize your playbooks. It helps make your automation scripts more flexible and reusable.

## Creating an Ansible inventory

The following list defines the tools and definitions that are used to define an inventory file:

- Configuration file** Ansible requires an `ansible.cfg` file to specify the default inventory location. By default, it searches for an `inventory` file in the current directory. You can customize this location according to your project structure and needs.
- Inventory file** The inventory file itself is the heart of your inventory. It can be on your Ansible control node or be fetched dynamically from external sources like cloud providers, external databases, or even scripts.
- Inventory structure** It is important to understand and adhere to the inventory structure. Whether you choose the INI, YAML, or plain text format, organizing hosts and groups correctly improves the readability and maintainability of your inventory.

## Using an Ansible inventory

There are several advantages of using an inventory in your Ansible environment:

- ▶ **Defining the infrastructure**

The inventory file provides a high-level view of your infrastructure by listing all the servers and their associated attributes.
- ▶ **Targeting specific hosts or groups**

By using the host and group information, you can run Ansible tasks on subsets of machines rather than applying them system-wide, which achieves more granular control over your automation.
- ▶ **Modifying host variables**

By using inventory files, you can define host-specific variables such as IP addresses, credentials, or package versions. These variables can be accessed within playbooks and used to customize behavior based on specific hosts.

Ansible inventory plays a crucial role in managing and automating your IT infrastructure. It helps Ansible identify the target systems, organize them into groups, define variables, and run tasks. The inventory file empowers you to automate tasks based on your specific needs by providing a clear and structured overview of your environment. Whether you are a system administrator, a DevOps engineer, or curious about automation, mastering the Ansible inventory is an essential skill to have in your toolkit.

## Defining inventory hosts and groups

Your inventory defines the managed nodes that you automate. Groups help you run automation tasks on multiple hosts concurrently. When your inventory is defined, you can use patterns to select the hosts or groups that you want Ansible to run against.

The simplest inventory is a single file with a list of hosts and groups. The default location for this file is `/etc/ansible/hosts`, but you can specify a different inventory file at the CLI by using the `-i <path>` option, or in the configuration file by using `inventory`.

Ansible [inventory plug-ins](#) support a range of formats and sources to make your inventory flexible and customizable. As your inventory expands, you might need more than a single file to organize your hosts and groups. Here are three options beyond the `/etc/ansible/hosts` file:

- ▶ You can create a directory with multiple inventory files, as described in “Organizing inventory in a directory” on page 70. They can use different formats (YAML, ini, and so on).
- ▶ You can pull inventory dynamically. For example, you can use a dynamic inventory plug-in to list resources in one or more cloud providers. For more information, see 2.3.2, “Overview of dynamic inventory” on page 70.
- ▶ You can use multiple sources for inventory, including both dynamic inventory and static files.

### Organizing inventory in a directory

You can consolidate multiple inventory sources in a single directory. The simplest version is a directory with multiple files instead of a single inventory file. A single file is too difficult to maintain when it gets too long. If you have multiple teams and multiple automation projects, having one inventory file per team or project lets everyone find the hosts and groups that matter to them.

You can also combine multiple inventory source types in an inventory directory, which can be useful for combining static and dynamic hosts and managing them as one inventory. The inventory directory that is shown in Example 2-6 combines an inventory plug-in source, a dynamic inventory script, and a file with static hosts.

*Example 2-6 Inventory directory with mixed sources*

---

```
inventory/  
  openstack.yml      # configure inventory plug-in to get hosts from OpenStack  
cloud  
  dynamic-inventory.py # add extra hosts with dynamic inventory script  
  on-prem            # add static hosts and groups  
  parent-groups      # add static hosts and groups
```

---

You can target this inventory directory as follows:

```
ansible-playbook example.yml -i inventory
```

You can also configure the inventory directory in your `ansible.cfg` file.

## 2.3.2 Overview of dynamic inventory

If you know in advance which servers that you want to manage by using Ansible, you can use a static inventory. However, sometimes the infrastructure is so dynamic that you cannot know the names of servers, or you might know the names but it is more efficient to get them from some other source. For example, you might want to perform some operations on Virtual I/O Servers (VIOSs) on a specific managed system, or run some playbooks on all logical partitions (LPARs) that PowerVC manages. This use case is the one for dynamic inventories.

A dynamic inventory defines the list of servers to manage by using a special inventory plug-in. You can find the list of plug-ins that are available to you by using the `ansible-doc` command, as shown in Example 2-7 on page 71.

*Example 2-7 List of plug-ins that are provided by the ansible-doc command*

---

```
# ansible-doc -t inventory -l
ansible.builtin.advanced_host_list Parses a 'host list' with ranges
ansible.builtin.auto                Loads and runs an inventory plug-in specified
in a YAML config
ansible.builtin.constructed        Uses Jinja2 to construct vars and groups based
on existing inventory
ansible.builtin.generator          Uses Jinja2 to construct hosts and groups from
patterns
ansible.builtin.host_list          Parses a 'host list' string
ansible.builtin.ini                Uses an Ansible INI file as an inventory source
ansible.builtin.script             Runs an inventory script that returns JSON
ansible.builtin.toml               Uses a specific TOML file as an inventory
source
ansible.builtin.yaml               Uses a specific YAML file as an inventory
source
ibm.power_hmc.powervm_inventory    HMC-based inventory source for Power Servers
openstack.cloud.openstack          OpenStack inventory source
```

---

Several inventory plug-ins are delivered with Ansible. They start with `ansible.builtin`. In this particular case, there are two more plug-ins that are provided by installed collections:

- ▶ `ibm.power_hmc.powervm_inventory`, which is provided by the `ibm.power_hmc` collection
- ▶ `openstack.cloud.openstack`, which is provided by the `openstack.cloud` collection.

As with static inventories, you can use the `ansible-inventory` command to test and show your inventory. In dynamic inventories, test your configuration before you run your playbook, as shown in Example 2-8.

*Example 2-8 Inventory test by using the ansible-inventory command*

---

```
# ansible-inventory -i hmc1.power_hmc.yml --list
{
  "_meta": {
    "hostvars": {
      "aixlpar1": {
        "ansible_host": "10.17.19.42"
      },
      "aixlpar2": {
        "ansible_host": "10.17.19.100"
      },
      "vio1": {
        "ansible_host": "10.17.19.113"
      },
      "vio2": {
        "ansible_host": "10.17.19.114"
      },
      "linux": {
        "ansible_host": "10.17.19.13"
      }
    }
  },
  "all": {
    "children": [
      "ungrouped",
      "P10-9080-HEX"
    ]
  }
}
```

```

    ]
  },
  "P10-9080-HEX": {
    "hosts": [
      "aixlpar1",
      "aixlpar2",
      "vio1",
      "vio2",
      "linux"
    ]
  }
}

```

---

From the output in Example 2-8 on page 71, you see that the dynamic inventory that was defined by using the `hmc1.power_hmc.yml` file found several AIX, Linux, and VIOS LPARs.

If you look in the configuration file, you find information about how to connect to the HMC and filters that determine which systems and LPARs that you see in the output, as shown in Example 2-9.

*Example 2-9 Contents of `hmc1.power_hmc.yml`*

---

```

# cat hmc1.power_hmc.yml
plug-in: ibm.power_hmc.powervm_inventory
hmc_hosts:
  - hmc: hmc1
    user: hscroot
    password: abcd1234
system_filters:
  SystemName: 'P10-9080-HEX'
filters:
  PartitionState: 'running'

```

---

Example 2-10 shows a common situation where you want to get a list of LPARs directly from an HMC. By using `powervm_inventory` from the `ibm.power_hmc` collection, you can dynamically define variables and assign servers to groups.

*Example 2-10 Defining variables dynamically by using `powervm_inventory`*

---

```

# cat hmc1.power_hmc.yml
plug-in: ibm.power_hmc.powervm_inventory
strict: False
hmc_hosts:
  - hmc: hmc1
    user: hscroot
    password: abcd1234
system_filters:
  SystemName: 'P10-9080-HEX'
filters:
  PartitionState: 'running'
compose:
  ansible_host: PartitionName
  ansible_python_interpreter: "'/opt/freeware/bin/python3' if 'AIX' in
OperatingSystemVersion or 'VIOS' in OperatingSystemVersion else
'/QOpenSys/pkgs/bin/python3' if 'IBM i' in OperatingSystemVersion else
'/usr/bin/python3'"

```



```

ansible_user: "'root' if 'AIX' in OperatingSystemVersion or 'Linux' in
OperatingSystemVersion else 'ansible' if 'VIOS' in OperatingSystemVersion else
'qsecofr'"
groups:
  AIX: "'AIX' in OperatingSystemVersion"
  Linux: "'Linux' in OperatingSystemVersion"
  IBM i: "'IBM i' in OperatingSystemVersion"
  VIOS: "'VIOS' in OperatingSystemVersion"

```

---

In this example, we define groups of hosts according to their operating systems (OSs). All VIOSs are assigned to the group VIOS. All IBM i LPARs are assigned to the group 'IBM i'. All Linux LPARs are assigned to the group Linux. All AIX LPARs are assigned to the group AIX.

We dynamically define variables for our hosts. We want to connect to AIX and Linux LPARs as user root and to IBM i LPARs as user qsecofr. We set the variable `<ansible_user>` depending on the LPAR's OS. We change the path to the Python interpreter based on the LPAR's OS.

If you use IBM PowerVC to manage your LPARs, you can use the `openstack.cloud.openstack` inventory plug-in to get the list of the LPARs in PowerVC. Use the inventory plug-in to set variables from `/opt/ibm/powervc/powervcrc` and then create an inventory file with one line in it, as shown in Example 2-11.

*Example 2-11 Using the `openstack.cloud.openstack` plug-in*

```

# cat openstack.yml
plug-in: openstack.cloud.openstack
# source /opt/ibm/powervc/powervcrc
# ansible-inventory -i openstack.yml --list

```

---

If you do not want to set environment variables, you can create a file that is named `clouds.yml` in the same directory with authentication parameters, as shown in Example 2-12. The file is used by the OpenStack inventory plug-in automatically.

*Example 2-12 Using `clouds.yml`*

```

# cat clouds.yml
clouds:
  powervc:
    identity_api_version: "3"
    region_name: RegionOne
    verify: false
    auth:
      auth_url: https://powervc:5000/v3
      project_name: "ibm-default"
      username: root
      password: ibmaix
      project_domain_name: Default
      user_domain_name: Default

```

---

You can get more information about the inventory plug-ins by using the `ansible-doc` command:

```

# ansible-doc -t inventory ibm.power_hmc.powervm_inventory
# ansible-doc -t inventory openstack.cloud.openstack

```

It is also possible to extend the current playbook with newly provisioned hosts by using a dynamic inventory.

Example 2-13 shows a playbook that consists of two plays, where one play targets the localhost and the second play targets the dynamic group "powervms".

*Example 2-13 Playbook targeting the localhost and dynamic group*

---

```
---
- hosts: localhost
  tasks:
    - name: create dynamic in memory inventory
      add_host:
        name: "{{ item.ip }}"
        groups: powervms
        ansible_connection: local
      loop:
        - ip: 1.2.3.4
          name: vm01
        - ip: 2.4.5.6
          name: vm02
        - ip: 3.4.5.6
          name: vm03
- hosts: powervms
  tasks:
    - name: wait for reachability
      wait_for_connection:
        delay: 5
        timeout: 240
    - name: gather_facts
      setup:
        gather_subset: min
    - debug:
        msg: "{{ ansible_hostname }} {{ansible_default_ipv4.address}}"
```

---

For documentation reasons, we set `ansible_connection: local`. (You may omit the line in production.) The output from Example 2-13 is shown in Example 2-14.

*Example 2-14 Playbook output*

---

```
WARNING]: the provided hosts list is empty, only localhost is available. The
implicit localhost does not match 'all'

PLAY [localhost]
*****
*****

TASK [create dynamic in memory inventory]
*****
Saturday 30 September 2023 14:02:08 +0200 (0:00:00.030) 0:00:00.030 ****
Saturday 30 September 2023 14:02:08 +0200 (0:00:00.030) 0:00:00.030 ****
changed: [localhost] => (item={'ip': '1.2.3.4', 'name': 'vm01'}) => {"add_host":
{"groups": ["powervms"], "host_name": "1.2.3.4", "host_vars":
{"ansible_connection": "local"}}, "ansible_loop_var": "item", "changed": true,
"item": {"ip": "1.2.3.4", "name": "vm01"}}
changed: [localhost] => (item={'ip': '2.4.5.6', 'name': 'vm02'}) => {"add_host":
{"groups": ["powervms"], "host_name": "2.4.5.6", "host_vars":
{"ansible_connection": "local"}}, "ansible_loop_var": "item", "changed": true,
"item": {"ip": "2.4.5.6", "name": "vm02"}}
```

```
changed: [localhost] => (item={'ip': '3.4.5.6', 'name': 'vm03'}) => {"add_host":
{"groups": ["powervms"], "host_name": "3.4.5.6", "host_vars":
{"ansible_connection": "local"}}, "ansible_loop_var": "item", "changed": true,
"item": {"ip": "3.4.5.6", "name": "vm03"}}
```

PLAY [powervms]

```
*****
*****
```

TASK [wait for reachability]

```
*****
```

```
Saturday 30 September 2023 14:02:08 +0200 (0:00:00.085) 0:00:00.116 ****
```

```
Saturday 30 September 2023 14:02:08 +0200 (0:00:00.085) 0:00:00.116 ****
```

```
[WARNING]: Reset is not implemented for this connection
```

```
[WARNING]: Reset is not implemented for this connection
```

```
[WARNING]: Reset is not implemented for this connection
```

```
ok: [3.4.5.6] => {"changed": false, "elapsed": 5}
```

```
ok: [1.2.3.4] => {"changed": false, "elapsed": 5}
```

```
ok: [2.4.5.6] => {"changed": false, "elapsed": 5}
```

TASK [gather\_facts]

```
*****
```

```
*****
```

```
Saturday 30 September 2023 14:02:13 +0200 (0:00:05.427) 0:00:05.544 ****
```

```
Saturday 30 September 2023 14:02:13 +0200 (0:00:05.427) 0:00:05.543 ****
```

```
ok: [2.4.5.6]
```

```
ok: [3.4.5.6]
```

```
ok: [1.2.3.4]
```

TASK [debug]

```
*****
```

```
*****
```

```
Saturday 30 September 2023 14:02:14 +0200 (0:00:00.805) 0:00:06.349 ****
```

```
Saturday 30 September 2023 14:02:14 +0200 (0:00:00.805) 0:00:06.348 ****
```

```
ok: [1.2.3.4] => {
  "msg": "vm9810 192.168.98.10"
```

```
}
```

```
ok: [2.4.5.6] => {
  "msg": "vm9810 192.168.98.10"
```

```
}
```

```
ok: [3.4.5.6] => {
  "msg": "vm9810 192.168.98.10"
```

```
}
```

PLAY RECAP

```
*****
```

```
*****
```

```
1.2.3.4 : ok=3 changed=0 unreachable=0 failed=0
```

```
skipped=0 rescued=0 ignored=0
```

```
2.4.5.6 : ok=3 changed=0 unreachable=0 failed=0
```

```
skipped=0 rescued=0 ignored=0
```

```
3.4.5.6 : ok=3 changed=0 unreachable=0 failed=0
```

```
skipped=0 rescued=0 ignored=0
```

```
localhost          : ok=1   changed=1   unreachable=0   failed=0
skipped=0   rescued=0   ignored=0
```

---

If you want to change a local inventory file instead of using an in-memory inventory, use the `lineinfile` or `template` module to populate the inventory file. To make Ansible use the most current file, there is a meta directive that rereads and reloads the inventory.

Example 2-15 shows a playbook that uses these two actions.

*Example 2-15 Creating a dynamic inventory file*

---

```
---
- hosts: localhost

  tasks:
    - name: create dynamic inventory
      local_action:
        module: lineinfile
        path: inventories/powervms.ini
        regexp: ^{{ item.name }}
        insertafter: "[powervms]"
        line: "{{ item.name }} ansible_host={{ item.ip }}"
      ansible_connection=local"
      loop:
        - ip: 1.2.3.4
          name: vm01
        - ip: 2.4.5.6
          name: vm02
        - ip: 3.4.5.6
          name: vm03

    - meta: refresh_inventory

- hosts: powervms

  tasks:
    - name: wait for reachability
      wait_for_connection:
        delay: 5
        timeout: 240

    - name: gather_facts
      setup:
        gather_subset: min

    - debug:
        msg: "{{ ansible_hostname }}" "{{ansible_default_ipv4.address}}"
```

---

Example 2-16 shows the output from the playbook in Example 2-15 on page 76.

*Example 2-16 Output of 'ansible-playbook -i playbooks/inventories/powervms.ini*

```
playbooks/dynamic2.yml -v --diff''
Script started, file is /tmp/a
Using /ansible/ALL/ansible/ansible.cfg as a config file
[WARNING]: * Failed to parse
/ansible/ALL/ansible/playbooks/inventories/powervms.ini with ini plug-in:
/ansible/ALL/ansible/playbooks/inventories/powervms.ini:2: Expected key=value host
variable assignment, got:
1.2.3.4
[WARNING]: Unable to parse /ansible/ALL/ansible/playbooks/inventories/powervms.ini
as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: The provided hosts list is empty, only localhost is available. The
implicit localhost does not match 'all'

PLAY [localhost]
*****
*****

TASK [create dynamic inventory]
*****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.041)      0:00:00.041 ****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.040)      0:00:00.040 ****
--- before: inventories/powervms.ini (content)
+++ after: inventories/powervms.ini (content)
@@ -1,4 +1,4 @@
 [powervms]
-vm01 1.2.3.4 ansible_connection=local
+vm01 ansible_host=1.2.3.4 ansible_connection=local
  vm02 2.4.5.6 ansible_connection=local
  vm03 3.4.5.6 ansible_connection=local

changed: [localhost] => (item={'ip': '1.2.3.4', 'name': 'vm01'}) =>
{"ansible_loop_var": "item", "backup": "", "changed": true, "item": {"ip":
"1.2.3.4", "name": "vm01"}, "msg": "line replaced"}
--- before: inventories/powervms.ini (content)
+++ after: inventories/powervms.ini (content)
@@ -1,4 +1,4 @@
 [powervms]
  vm01 ansible_host=1.2.3.4 ansible_connection=local
-vm02 2.4.5.6 ansible_connection=local
+vm02 ansible_host=2.4.5.6 ansible_connection=local
  vm03 3.4.5.6 ansible_connection=local

changed: [localhost] => (item={'ip': '2.4.5.6', 'name': 'vm02'}) =>
{"ansible_loop_var": "item", "backup": "", "changed": true, "item": {"ip":
"2.4.5.6", "name": "vm02"}, "msg": "line replaced"}
--- before: inventories/powervms.ini (content)
+++ after: inventories/powervms.ini (content)
@@ -1,4 +1,4 @@
 [powervms]
  vm01 ansible_host=1.2.3.4 ansible_connection=local
  vm02 ansible_host=2.4.5.6 ansible_connection=local
```

```

-vm03 3.4.5.6 ansible_connection=local
+vm03 ansible_host=3.4.5.6 ansible_connection=local

changed: [localhost] => (item={'ip': '3.4.5.6', 'name': 'vm03'}) =>
{"ansible_loop_var": "item", "backup": "", "changed": true, "item": {"ip":
"3.4.5.6", "name": "vm03"}, "msg": "line replaced"}

TASK [meta]
*****
*****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.709)      0:00:00.750 ****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.709)      0:00:00.750 ****

PLAY [powervms]
*****
*****

TASK [Gathering Facts]
*****
*****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.041)      0:00:00.792 ****
Saturday 30 September 2023  14:14:36 +0200 (0:00:00.041)      0:00:00.791 ****
ok: [vm03]
ok: [vm02]
ok: [vm01]

TASK [wait for reachability]
*****
Saturday 30 September 2023  14:14:38 +0200 (0:00:01.386)      0:00:02.179 ****
Saturday 30 September 2023  14:14:38 +0200 (0:00:01.386)      0:00:02.178 ****
[WARNING]: Reset is not implemented for this connection
[WARNING]: Reset is not implemented for this connection
[WARNING]: Reset is not implemented for this connection
ok: [vm02] => {"changed": false, "elapsed": 5}
ok: [vm03] => {"changed": false, "elapsed": 5}
ok: [vm01] => {"changed": false, "elapsed": 5}

TASK [gather_facts]
*****
*****
Saturday 30 September 2023  14:14:43 +0200 (0:00:05.398)      0:00:07.577 ****
Saturday 30 September 2023  14:14:43 +0200 (0:00:05.398)      0:00:07.576 ****
ok: [vm03]
ok: [vm02]
ok: [vm01]

TASK [debug]
*****
*****
Saturday 30 September 2023  14:14:44 +0200 (0:00:00.541)      0:00:08.118 ****
Saturday 30 September 2023  14:14:44 +0200 (0:00:00.541)      0:00:08.117 ****
ok: [vm01] => {
    "msg": "vm9810 192.168.98.10"
}
ok: [vm02] => {

```

```

    "msg": "vm9810 192.168.98.10"
  }
ok: [vm03] => {
    "msg": "vm9810 192.168.98.10"
  }

```

PLAY RECAP

```

*****
*****

```

localhost		: ok=1	changed=1	unreachable=0	failed=0
skipped=0	rescued=0	ignored=0			
vm01		: ok=4	changed=0	unreachable=0	failed=0
skipped=0	rescued=0	ignored=0			
vm02		: ok=4	changed=0	unreachable=0	failed=0
skipped=0	rescued=0	ignored=0			
vm03		: ok=4	changed=0	unreachable=0	failed=0
skipped=0	rescued=0	ignored=0			

---

## 2.4 Ansible tasks, playbooks, and modules

In Ansible, a playbook consists of one or more plays in an ordered list. Each play runs part of the overall goal of the playbook by running one or more tasks. A task is the smallest unit of work that is automated by using the playbook.

Each task uses Ansible modules to achieve expected outcomes. A module is a reusable, stand-alone script that Ansible runs, either locally or remotely. Modules interact with the local machine, an API, or a remote system to perform specific tasks like changing a database password or starting a cloud instance.

### 2.4.1 Creating Ansible playbooks

Ansible uses the YAML syntax, and playbooks are expressed in YAML format with a minimum of syntax. If you are not familiar with YAML, see 2.2, “Understanding the Ansible declarative language” on page 64. For more information, see [YAML Syntax](#).

Consider installing an add-on for your text editor to help you write clean YAML syntax in your playbooks. If your preferred editor is `vi` (or `vim`), then you may add these lines to `~/ .vimrc` to use TAB by making the TAB character appear as 2 blank spaces:

```

autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab

```

For more information about setting up `vim`, see [Setup Your vim editor for Ansible Playbook](#).

## Anatomy of an Ansible playbook

A *playbook* is a text file that is written in YAML format, and is normally saved with the extension `.yml` or `.yaml`.

A playbook is what drives Ansible automation. The following concepts are important to understand when building playbooks for your environment:

<b>Playbook</b>	A text file that contains a list of one or more plays to run in a specific order, from top to bottom, to achieve an overall goal.
<b>Play</b>	An ordered list of tasks that maps to managed nodes in an inventory. It is the top-level specification for a group of tasks. Defined in the play are the hosts that it runs on (the inventory) and control behaviors, such as fact-gathering or privilege level. Multiple plays can exist within a single Ansible playbook and may run on different hosts.
<b>Task</b>	The application of a module to perform a specific unit of work. A play combines a sequence of tasks that are applied, in order, to one or more hosts that are selected from your inventory.
<b>Module</b>	Parametrized components or programs with internal logic, representing a single step to be done on the target machine. The modules “do” things in Ansible.
<b>Plug-ins</b>	Pieces of code that augment Ansible core functions. They are often provided by a manufacturer for their specific devices. Ansible uses a plug-in architecture to enable a rich, flexible, and expandable feature set.

The playbook uses indentation with space characters to indicate the structure of its data. YAML does not place strict requirements on how many spaces are used for the indentation, but there are two basic rules:

- ▶ Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation.
- ▶ Items that are children of another item must be indented more than their parents.

You can also add blank lines for readability.

**Note:** Only the space character can be used for indentation. TAB characters are not allowed.

## Start of a playbook

A playbook starts with a line consisting of three dashes (`---`) as a starting document marker and may end with three dots (`...`) as an end-of-document marker (the `...` is optional and in practice is often omitted).

Between these markers, the playbook contains a list of plays. Each item in a YAML list starts with a single dash followed by a space. Example 2-17 shows an example playbook that is designed to capture the `oslevel` from the system.

*Example 2-17 Sample playbook capturing the `oslevel`*

---

```
---
- name: GET oslevel AIX
  hosts: all

  tasks:
```



```
- name: Gather LPP Facts
  shell: "oslevel -s"
  register: output_oslevel

- name: Print the oslevel
  debug:
    msg: "{{ ansible_hostname }}" has the AIX oslevel of {{
output_oslevel.stdout }}"
```

---

## Order in plays

The order in plays is always the following one:

1. `pre_tasks`:
2. `roles`:
3. `tasks`:
4. `handlers`:

You may change the order by using `include_roles`, `import_roles`, `include_tasks`, or `import_tasks`. You can also use the directive `tasks_from`: while including tasks.

The main differences between `import` and `include` are the following ones:

- ▶ All `import*` statements are preprocessed at the time that playbooks are parsed.
- ▶ All `include*` statements are processed as they are encountered during the run of the playbook.

So, `import` is static, and `include` is dynamic.

A best practice is to use `import` when you deal with logical “units”. For example, separate long list of tasks into subtask files in a `main.yml` file. The `include` keyword is used to make decisions based on dynamically gathered facts, as shown here:

```
- include_tasks: taskrun_{{ ansible_os_family | lower }}.yml
```

For more information about using `import` or `include`, see “Including and importing other playbooks” on page 83.

## Verifying playbooks

You might want to verify your playbooks to detect syntax errors and other problems before you run them. The `ansible-playbook` command offers several options for verification, including `--check`, `--diff`, `--list-hosts`, `--list-tasks`, and `--syntax-check`. To check the playbook for syntax errors by using `--syntax-check`, use the command that is shown in Example 2-18.

*Example 2-18 Verifying a playbook*

---

```
$ ansible-playbook --syntax-check aix_oslevel.yml
playbook: aix_oslevel.yml
```

---

## Ansible tasks

Ansible tasks form the core building blocks of automation that you can use to specify the state of your infrastructure and applications in a declarative manner.

Figure 2-2 shows how tasks relate to the rest of the tools within the Ansible ecosystem.

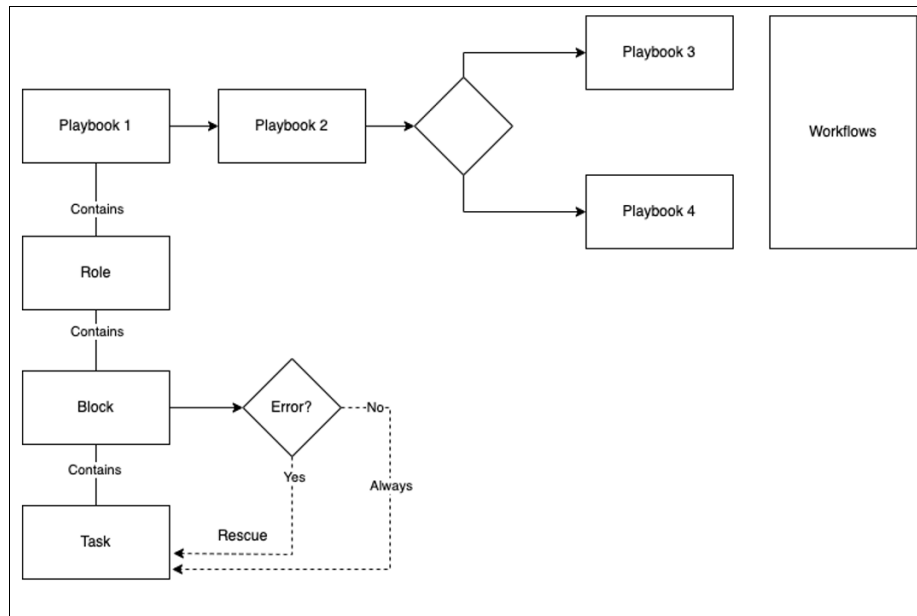


Figure 2-2 Task flow chart

Tasks are fundamental building blocks in Ansible, and there is a plethora of plug-ins that are available. Each plug-in is further divided into modules and grouped as collections. The plug-ins are in different categories, and Ansible categorizes the plug-ins based on what they do. For example, the `become` plug-in enables you to become a superuser or run the function as a specific user.

The plug-in ecosystem is vast and includes Ansible built-in plug-ins, community plug-ins from contributors, and vendor plug-ins from companies such as Cisco, IBM, and AWS. This [Ansible document](#) provides a list of plug-ins and their modules.

### Writing tasks

Each task in Ansible consists of three components that are defined in YAML format.

The task starts with a name (in free-form text) to describe what is being done. It is a best practice to provide a good description that matches the task.

Then, you call a plug-in, which has the format `<namespace.plugin.module>`, which is followed by its arguments or parameters.

Example 2-19 shows `get_url`, which is a built-in plug-in that is in the Ansible namespace. It is an example playbook for IBM AIX that downloads and configures the Yellowdog Updater, Modified (YUM) package manager, and then installs MariaDB.

For Ansible built-in plug-ins, you may omit the namespace (`ansible`) and plug-in (`builtin`). The `get_url` also shows the parameters that you need to pass, such as `url`, `dest`, `mode`, and `validate_certs`.

#### Example 2-19 The `get_url` plug-in

```
---
- name: Install MariaDB open source relational database
  hosts: ansible-vm
  tasks:
```

```

- name: Download 'yum.sh' script
  get_url:
    url:
      https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/yum.sh
    dest: /tmp/yum.sh
    mode: 0755
    validate_certs: False

- name: Run the 'yum.sh' script
  shell: /tmp/yum.sh

- name: Install MariaDB package
  yum:
    name: mariadb-server
    state: latest

```

---

### ***Including and importing other playbooks***

Ansible tasks can include or import other tasks that were prebuilt. The difference between import and include is subtle, but critical to understand. The `import_tasks` are preprocessed when Ansible processes the playbook for running. The `include_tasks` are processed when the playbook runs.

Example 2-20 shows an excerpt from a playbook that shows the usage of both `import_tasks` and `include_tasks`.

#### *Example 2-20 Playbook with the import and include tasks*

```

- name: Create a file system when disks_configuration variables is a dictionary
  import_tasks: disks-dict2list.yml
  when: disks_configuration | type_debug == "dict"

- name: Create a file system when disks_configuration variables is a list
  include_tasks: file-system-creation-core.yml
  with_items:
    - "{{ disks_configuration }}"
  loop_control:
    loop_var: file_system
  when: disks_configuration | type_debug == "list"

```

---

It is a best practice to include tasks that apply to a specific target OS or target hardware. To get better readability of playbooks, you may use the pattern that is shown in Example 2-21.

#### *Example 2-21 Using variables to target specific target attributes*

```

name: get information of underlying play_hosts
  setup:
    gather_subset: min
    tags: vars
- name: gather os specific variables
  include_vars: "{{ item }}"
  with_first_found:
    - "{{ ansible_distribution }}-{{ ansible_distribution_major_version }}.yml"
    - "{{ ansible_distribution }}.yml"
  tags: vars

```

---

## Variables

Ansible uses variables to manage differences between systems. With Ansible, you can run tasks and playbooks on multiple different systems with a single command. To represent the variations among those different systems, you can create variables with standard YAML syntax, including lists and dictionaries. You can define these variables in your playbooks, in your inventory, in re-usable files or roles, or at the CLI. You can also create variables during a playbook run time by registering the return value or values of a task as a new variable.

### Defining variables in playbooks

The simplest way to define variables is to put a vars section in your playbook with the names and values of your variables, as shown in Example 2-22.

Example 2-22 Defining variables

```
vars:
  tls_dir: /etc/nginx/ssl/
  key_file: nginx.key
  cert_file: nginx.crt
  conf_file: /etc/nginx/sites-available/default
  server_name: localhost
  firewall_pkg: firewalld
  firewall_svc: firewalld
  web_pkg: httpd
  web_svc: httpd
```

Not all strings are valid Ansible variable names. A variable name can include only letters, numbers, and underscores. Python keywords or playbook keywords are not valid variable names. A variable name cannot begin with a number.

Variable names can begin with an underscore. In many programming languages, variables that begin with an underscore are private, which is not true in Ansible. Variables that begin with an underscore are treated the same as any other variable. Do not rely on this convention for privacy or security.

Figure 2-3 gives examples of valid and invalid variable names.

Valid variable names	Not valid
<code>foo</code>	<code>*foo</code> , Python keywords such as <code>async</code> and <code>lambda</code>
<code>foo_env</code>	playbook keywords such as <code>environment</code>
<code>foo_port</code>	<code>foo-port</code> , <code>foo port</code> , <code>foo.port</code>
<code>foo5</code> , <code>_foo</code>	<code>5foo</code> , <code>12</code>

Figure 2-3 Valid and invalid Ansible variable names

### Defining variables in separate files

With Ansible, you can put variables into one or more files, which are then referenced in the playbook by using a section that is called `vars_files`. For example, you might want to take the preceding example and put the variables in a file that is named `nginx.yml` instead of putting them in the playbook. To do so, replace the `vars` section with a `vars_files` that looks like what is shown in Example 2-23 on page 85.

---

*Example 2-23 Defining variables in separate files*

---

```
vars_files:  
  - vars_nginx.yml
```

---

The vars\_nginx.yml file looks like Example 2-22 on page 84.

**Referencing variables**

After you define a variable, use Jinja2 syntax to reference it. Jinja2 variables use double curly braces. For example, the expression `My amp goes to {{ max_amp_value }}` demonstrates the most basic form of variable substitution. You can use Jinja2 syntax in playbooks, as shown in Example 2-24.

---

*Example 2-24 Referencing variables*

---

```
ansible.builtin.template:  
  src: foo.cfg.j2  
  dest: '{{ remote_install_path }}/foo.cfg'
```

---

When you want to display a debug message with a variable, use a double quotation mark string with the variable name embedded in double braces, as shown in Example 2-25.

---

*Example 2-25 Referencing variables*

---

```
- name: Display the variable  
  debug:  
    msg: "The file used was {{ conf_file }}"
```

---

Variables can be concatenated between the double braces by using the tilde operator `~`, as shown in Example 2-26.

---

*Example 2-26 Concatenating variables*

---

```
- name: Concatenate variables  
  debug:  
    msg: "Hello! Your URL is https://{{ server_name ~ '.' ~ domain_name }}/'"
```

---

**Registering variables**

You can create variables from the output of an Ansible task with the task keyword `register`. You can use registered variables in any later tasks in your play. Each Ansible module returns results in JSON format. To use these results, create a registered variable by using the `register` clause when starting a module, as shown in Example 2-27.

---

*Example 2-27 Registering variables*

---

```
- name: Get disk information  
  ansible.builtin.shell: |  
    fdisk -l  
    lsblk  
    df -h  
  register: os_disk  
  
- debug:  
  var: os_disk
```

---

Example 2-28 shows how to capture the output of the `whoami` command to a variable that is named `<logon>`.

*Example 2-28 Capturing the whoami output*

---

```
- name: Capture output of whoami command
  command: whoami
  register: logon
```

---

For more information about the usage of variables. see [playbooks\\_variables](#).

### Execution control

When you run Ansible tasks, the results are easily identified as successful or unsuccessful because by default they return green, yellow, or red:

- Green                      A task ran as expected, and no change was made.
- Yellow                    A task ran as expected and made a change.
- Red                        A task failed to run successfully.

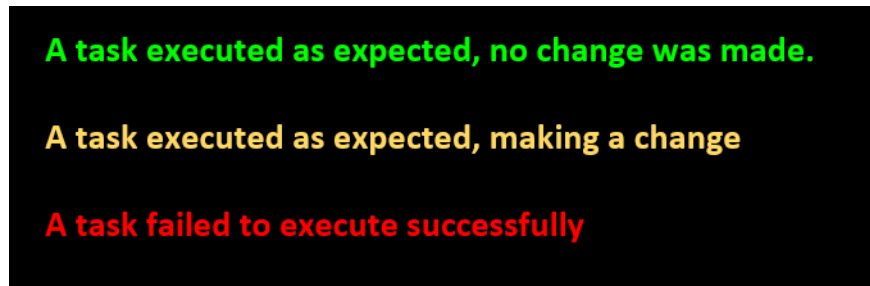


Figure 2-4 Ansible colors

## 2.5 Ansible roles and collections

Ansible roles are basically playbooks that are placed into a known file structure. Moving to roles from a playbook makes sharing, reading, and updating your Ansible workflow simpler. Users can write their own roles, and they are often provided with roles that are part of collections.

### 2.5.1 Understanding roles in Ansible

Ansible roles are a way to organize your playbooks into reusable units that can be shared across different projects or teams. A *role* is essentially a collection of tasks and other files that are related to a specific task or function within your infrastructure. For example, you can create a role to install and configure IBM WebSphere® Application Server on a remote server, and then use that role in different playbooks that deploy different web applications.

To create a role, follow a standard directory structure with eight main directories: `tasks`, `handlers`, `vars`, `defaults`, `meta`, `library`, `module_utils`, and `lookup_plugins`. Each directory contains a `main.yml` file that holds the relevant content for that directory. For example, the `tasks/main.yml` file contains the main list of tasks that the role runs, and the `vars/main.yml` file contains the variables that are associated with the role. You can also use other files and directories within each directory as needed.

To use a role in a playbook, you can either include it at the play level by using the `roles` keyword, or import it at the task level by using the `import_role` or `include_role` modules. You can also pass parameters to the role by using the `vars` keyword or the `args` keyword. Also, you can specify role dependencies in the `meta/main.yml` file, which means that Ansible automatically runs those roles before the current role.

Ansible roles are a powerful feature that can help in your configuration management and automate your deployments. You can also use Ansible Galaxy to find and share roles that other users create. For more information, see [Ansible Roles](#).

## 2.5.2 Creating and structuring Ansible roles

This par describes the process of creating and structuring Ansible roles by using two distinct resources: the official Ansible documentation and insights from Red Hat.

- ▶ From the Ansible Documentation

Creating Ansible roles involves a structured approach that aligns with best practices. The official Ansible documentation offers a comprehensive guide to creating roles that are intuitive, organized, and simple to maintain. This resource provides a in-depth look into the directory structure that forms the backbone of a well-structured role. To explore this approach, see [Official Ansible Role Directory Structure](#).

- ▶ The Red Hat perspective

Red Hat, a pioneer in the field of open-source technology, provides its own insights into developing Ansible roles. Their approach offers a practical and hands-on perspective that complements the official documentation. By adhering to the Red Hat methodology, you can gain a clearer understanding about how to develop Ansible roles effectively. For more information about a step-by-step guide on creating roles the Red Hat way, see [Developing Ansible Roles the Red Hat way](#).

### Crafting a role for IBM i virtual machine deletion

In this segment, we follow guidance to create a specialized Ansible role that is focused on deleting IBM i virtual machines (VMs). The process aligns with industry best practices, which helps ensure efficient, modular, and documented automation. By adhering to these principles, you become equipped to build a robust and effective Ansible role that is tailored for IBM i VM deletion.

Complete the following steps:

1. Creating an Ansible role begins with establishing a structured directory, which helps ensure clarity, ease of maintenance, and efficient collaboration. To initiate the process, run the following command (tailor the directory path to your environment):

```
ansible-galaxy init ~/itsoxx/itsoroles/delete_vm_via_powervc
```

That command sets the foundation for your role's directory structure, which adheres to best practices.

2. Inside the role's directory, there are key files like `defaults/main.yml`, `meta/main.yml`, and `tasks/main.yml`. These files define the role's configuration and behavior. By editing these files, you align your role with the needed functions. A core aspect of this step is refining the defaults values, role variables, and any specific instructions that are associated with your role.

3. In the `defaults/main.yml` file, define default values for the role's variables, which enhance flexibility and customization. Example 2-29 displays a snippet showcasing how variables are defined.

*Example 2-29 Defining default variables for role flexibility*

---

```
---
# Defaults value for deleting an IBM i VM
project: ibm-default
project_domain: Default
user_domain: Default
verify_cert: false
deploy_timeout: 300
availability_zone_name: 'Default Group'
...
```

---

4. The core tasks of your role are defined in the `tasks/main.yml` file, where the automation happens. Example 2-30 is an excerpt that shows the deletion of an IBM i VM by using the PowerVC module.

*Example 2-30 Performing core role tasks*

---

```
---
# Delete IBM i VM information from OpenStack
- name: Delete an IBM i VM
  os_server:
    auth:
      auth_url: https://{{ ansible_ssh_host }}:5000/v3
      username: '{{ ansible_ssh_user }}'
      password: '{{ ansible_ssh_pass }}'
      project_name: '{{ project }}'
      project_domain_name: '{{ project_domain }}'
      user_domain_name: '{{ user_domain }}'
    name: '{{ vm_name }}'
    verify: '{{ verify_cert }}'
    state: '{{vm_state}}'
    timeout: '{{ deploy_timeout }}'
  register: server_info
...
```

---

5. The `meta/main.yml` file is your role's calling card. It provides crucial metadata for users who can interact with your role. Example 2-31 is a snippet that showcases the role's author, description, company, supported platforms, and more.

*Example 2-31 Role metadata and description*

---

```
galaxy_info:
  author: Marcelo Avalos Del Carpio
  description: Ansible role to delete an IBM i VM through PowerVC
  company: IBM
  license: Apache-2.0
  min_ansible_version: 2.9
  platforms:
    - name: IBM i
      versions:
        - 7.2
        - 7.3
```



```
- 7.4
galaxy_tags:
- powervc
- ibmi
```

---

6. A documented role is a valuable asset. The README.md file provides clear instructions about how to use your role, its variables, and associated tasks. Example 2-32 is a snippet that demonstrates the structure and content of a comprehensive readme file.

*Example 2-32 Comprehensive readme file structure*

---

```
delete_vm_via_powervc
=====
The Ansible role to delete an IBM i VM through PowerVC.

Role Variables
-----

| Variable | Type | Description |
|-----|-----|-----|
| `vm_name` | str | Required. Name of the deployed vm. |
| `vm_state` | str | Action to perform (present/absent) |

Example Playbooks
-----
...
---
- name: Delete a vm
  hosts: powervc
  tasks:
  - include_role:
    name: delete_vm_via_powervc
  vars:
    vm_name: 'itso0x'
    vm_state: 'absent'
...
---
License
-----

Apache-2.0
```

---

**Note:** When publishing the created role, it is important to adhere to the formatting guidelines for the README.md file. Using triple backticks is crucial to help ensure proper readability on GitHub. For more information, see [GitHub Code Blocks](#), which documents creating and highlighting code blocks.

### 2.5.3 Sharing and reusing roles in multiple playbooks

One of the benefits of using Ansible is that you can create reusable artifacts that can be used in different scenarios and contexts. Roles are one of the ways to achieve this goal. A *role* is a collection of related tasks, variables, defaults, handlers, and other Ansible components that can be applied to a group of hosts or a specific playbook.

Roles organize your automation work into smaller, more manageable units that can be shared and reused. You can use roles to abstract common functions, such as installing a web server or updating a database, and then use them in multiple playbooks or even multiple times within one playbook.

To use roles in your playbooks, follow a defined directory structure that Ansible recognizes. Each role must have at least one of the following directories:

tasks:	The main list of tasks that the role runs.
handlers:	Handlers, which may be used within or outside this role.
library:	Modules, which may be used within this role.
files:	Files that the role deploys.
templates:	Templates that the role deploys.
vars:	Other variables for the role.
defaults:	Default variables for the role. These variables have the lowest priority of any variables that are available, and can be overridden by any other variable, including inventory variables.
meta:	Metadata for the role, including role dependencies.

Each directory must contain a `main.yml` file (or `main.yaml` or `main`) that contains the relevant content for that directory. You can also use other YAML files in some directories to organize your tasks or variables better.

**Note:** For more information about sharing and reusing roles across multiple playbooks, see the following resources:

- ▶ [Ansible Playbook Reuse Guide](#):
- ▶ [Sharing and Reusing Roles in Ansible](#)
- ▶ [Playbook Reuse with Ansible Roles](#)

## 2.5.4 Role dependencies and role-based variables

Role dependencies and role-based variables are important concepts in Ansible. This section explains what they are, how they work, and why they are useful.

### Role dependencies

Role dependencies automatically pull in other roles when you use a role, which helps ensure that the target computer is in a predictable condition before running your tasks. For example, you can have a common role that installs some packages and updates the system, and you want to run it before any other role.

To define role dependencies, create a `meta/main.yml` file inside your role directory with a `dependencies` block. Example 2-33 on page 91 shows an example of dependencies.

### Example 2-33 Role dependencies and conditional configuration in YAML

---

```
dependencies:
  - role: common
  - role: sshd
    enable_sshd: false
    when: environment == 'production'
```

---

Before running the current role, Ansible first runs the common role and then the sshd role, but only if the environment variable is set to 'production'. You can also pass variables to the dependent roles by using the same syntax.

Role dependencies run before the roles that depend on them, and they run once per playbook run. If two roles state the same one as their dependency, it is run only the first time.

For example, if you have three roles, role1, role2, and role3, and both role1 and role2 depend on role3, the run order is role3, role1, and role2.

You can override this behavior by setting `allow_duplicates: true` in the `meta/main.yml` file of the dependent roles, which makes Ansible run the role every time it is listed as a dependency. For example, if you set `allow_duplicates: true` for role3, the run order can be role3, role1, role3, and role2.

Role dependencies are a feature of Ansible that can help you reuse your roles and simplify your playbooks. However, use them with caution and avoid creating circular dependencies or complex dependency chains that can make your roles hard to maintain and debug.

**Note:** For more information about role dependencies, see the [Ansible Community Documentation](#) or the [tutorial](#).

## 2.5.5 Using collections

*Collections* are a way to package and distribute Ansible content, such as playbooks, roles, modules, and plug-ins. Collections can help you organize your Ansible projects and share them with other users or teams.

### What are collections

A collection is a directory of files that follows a specific structure and contains a `MANIFEST.json` file that defines its metadata. A collection can include any type of Ansible content, such as the following examples:

<b>Playbooks</b>	YAML files that define tasks to run on hosts.
<b>Roles</b>	Reusable units of Ansible content that can include tasks, variables, templates, files, and handlers.
<b>Modules</b>	Python files that run tasks on hosts and return information to Ansible.
<b>Plug-ins</b>	Python files that extend Ansible functions, such as lookup, filter, inventory, callback, and strategy plug-ins.

A collection can also have dependencies on other collections, which are specified in a `meta/requirements.yml` file. With this file, you can reuse existing content from other sources and avoid duplication.

## How to use collections

There are several ways to use collections in Ansible, depending on your needs and preferences. Here are some of the common methods:

- ▶ Installing collections from a distribution server: You can use the `ansible-galaxy` command to install collections from a server like Ansible Galaxy or a Pulp 3 Galaxy server. You can also use a requirements file to install multiple collections at once.
- ▶ Installing collections from source files: You can use the `ansible-galaxy` command to install collections from local source files, such as a Git repository or a `.tar` file.
- ▶ Using collections in a playbook: You can reference collection content by its fully qualified collection name (FQCN), which consists of the namespace, the collection name, and the content name. For example, `my_namespace.my_collection.my_module`. You can also use the `collections` keyword in your playbook or role to simplify the module names and avoid typing the FQCN every time.
- ▶ Using collections in roles: You can use the `collections` keyword in your role's `meta/main.yml` file to define which collections your role should search for unqualified module and action names.

## What are the benefits of collections

Collections offer several advantages for Ansible users and developers:

<b>Modularity</b>	With collections, you can group related content and separate it from other content to manage, maintain, and update your Ansible projects.
<b>Re-usability</b>	With collections, you can reuse existing content from other sources. You can also share your own collections with others through distribution servers or source control.
<b>Compatibility</b>	With collections, you can avoid conflicts between different versions of Ansible content. You can specify the minimum Ansible version and the required collections for your collection to work properly.
<b>Customization</b>	With collections, you can customize your Ansible environment by adding new modules and plug-ins that suit your needs. You can also override existing content by using a higher precedence collection.

**Note:** For more information, see the following resources:

- ▶ [Ansible User Guide on Using Collections](#)
- ▶ [Ansible Galaxy Guide on Creating Collections](#)
- ▶ [Ansible Community General Collection on GitHub](#)

## 2.6 Best practices for playbook and role design

When designing playbooks and roles, follow some best practices to help ensure the quality, readability, and maintainability of your code. Here are some suggestions:

- ▶ Use consistent naming conventions for your files, variables, tasks, and handlers. For more information, see the [Ansible documentation style guide](#).
- ▶ Use tags to group related tasks and allow for selective execution of your playbooks. For example, you can use tags to run only the tasks that are related to installing packages or configuring services.

- ▶ Use variables to store values that can change depending on the environment, such as hostnames, ports, and passwords. Avoid hardcoding these values in your tasks. Use the `ansible-vault` command to encrypt sensitive variables if needed.
- ▶ Use roles to organize your tasks into reusable units of functions. Roles can also include files, templates, variables, defaults, handlers, and meta information. Use the `ansible-galaxy` command to create and manage roles.
- ▶ Use `include` and `import` statements to modularize your playbooks and roles. `include` statements allow for dynamic inclusion of tasks or roles based on conditions or variables. `import` statements allow for static inclusion of files or roles at parse time.
- ▶ Use conditionals, loops, filters, and plug-ins to add logic and flexibility to your tasks. For example, you can use conditionals to check the state of a system before performing an action, loops to iterate over a list of items, filters to manipulate data, and plug-ins to extend the functions of Ansible.
- ▶ Use facts and registered variables to capture information from the hosts and use it in your tasks. Facts are variables that are automatically gathered by Ansible when running a playbook. Registered variables are variables that are created by registering the output of a task.
- ▶ Use error handling techniques to deal with failures and unexpected situations. For example, you can use the `ignore_errors`, `failed_when`, `changed_when`, and `rescue` or `always` keywords to control the behavior of your tasks when an error occurs.

## 2.6.1 Writing modular and reusable playbooks

One of the benefits of using Ansible is that you can use it to write playbooks that are modular and reusable. Therefore, you can avoid repeating the same tasks or variables in different playbooks, and instead use existing modules, roles, or `include` statements to reuse code. As a result, your playbooks are more maintainable, scalable, and reliable.

Here are some of the ways to write modular and reusable playbooks:

- Use modules** Modules are reusable pieces of code that perform specific tasks, such as installing packages, creating files, or managing services. Ansible has hundreds of built-in modules that you can use in your playbooks, or you can write your own custom modules. Modules can take parameters to customize their behavior, and return information that you can use in other tasks or templates.
- Use roles** Roles are collections of tasks, variables, files, templates, and handlers that are organized by a specific function or purpose. With roles, you can group related tasks and reuse them in multiple playbooks. Roles also support dependencies, which means that you can specify other roles that need to be run before or after a role.
- Use include statements** With `include` statements, you can dynamically load tasks, variables, handlers, or roles from another file or directory. Therefore, you can split your playbooks into smaller and more manageable files, and avoid duplication of code. You can also use conditional statements or loops to control when and how many times an `include` statement runs.

**Note:** For more information, see the following resources:

- ▶ [Ansible Documentation](#)
- ▶ [Ansible Best Practices](#)
- ▶ [Ansible Modules](#)

## 2.6.2 Using Ansible Galaxy for role management

Ansible Galaxy serves as a shared repository for Ansible roles and collections by offering prepackaged units of work that can be shared and reused within the Ansible community. This section describes the process of using Ansible Galaxy to discover, install, and employ roles and collections for your automation endeavors. Also, this section describes your own roles and collections and uploading them to the Galaxy platform.

### Discovering and installing roles and collections from Galaxy

To discover roles and collections on Galaxy, use the `ansible-galaxy search` command with filters, such as `author`, `tag`, `platform`, or `keyword`. For example, to locate roles pertaining to Linux by the author "itso," run the following command:

```
ansible-galaxy search linux --author itso --role
```

Example 2-34 displays a list of matching roles.

*Example 2-34 Matching roles in the Ansible Galaxy search results*

---

```
Found 2 roles matching your search:
Name                               Description
----                               -
itso.linux_common                  Common tasks for Linux on Power.
itso.linux_application             Application deployment on Linux.
```

---

To see more role details, run the following command:

```
ansible-galaxy info itso.linux_common
```

To install a role from Galaxy, run the following command:

```
ansible-galaxy install itso.linux_common
```

### Applying roles and collections in playbooks

To employ a role in a playbook, adding it to the `roles` section is sufficient, as shown in Example 2-35.

*Example 2-35 Implementing the `itso.linux_common` role in an Ansible playbook*

---

```
- hosts: all
  roles:
    - itso.linux_common
```

---

Variables can be passed by using the `vars` or `vars_files` keywords, as shown in Example 2-36.

*Example 2-36 Applying the `itso.linux_common` role with custom variables in an Ansible playbook*

---

```
- hosts: all
  roles:
    - role: itso.linux_common
      vars:
        var_name: value
```

---

Using collections in playbooks requires specifying the full namespace, as shown in Example 2-37.

*Example 2-37 Using the `community.general` module in an Ansible playbook task*

---

```
- hosts: all
  tasks:
    - name: Task that uses community.general module
      community.general.module_name:
        param1: value
        param2: value
```

---

## Uploading roles and collections to Galaxy

Creating a role begins with running `ansible-galaxy init` command:

```
ansible-galaxy init myrole
```

To create a collection, run the `ansible-galaxy collection init` command:

```
ansible-galaxy collection init mynamespace.mycollection
```

Editing files such as `readme` file and `metadata` is essential. Uploading requires an account on the Galaxy website and an API key. The building and uploading tasks use the commands that are shown in Example 2-38.

*Example 2-38 Building and publishing an Ansible role to Ansible Galaxy*

---

```
$ ansible-galaxy role build myrole
$ ansible-galaxy role publish myrole-1.0.0.tar.gz --api-key <API_KEY>
```

---

**Note:** Ansible Galaxy stands as a valuable asset for harnessing the potential of Ansible automation through its extensive collection of roles and collections. Empowered by the `ansible-galaxy` CLI tool, users can explore, install, and contribute to these automation components, which foster collaboration and efficiency. For more in-depth insights, see the following resources:

- ▶ [Ansible Galaxy's guide on creating roles](#)
- ▶ [Ansible Galaxy's user guide](#)

## 2.7 Creating versions and documenting playbooks and roles

One of the best practices for Ansible is to use roles to organize and reuse your automation content. Roles are self-contained units of Ansible automation that can be shared and used by multiple playbooks. Roles can also include custom modules, which are small programs that perform actions on remote hosts or on their behalf.

This section describes how to create versions of and document your playbooks and roles so that you can maintain them and collaborate with others effectively.

### 2.7.1 Creating versions of playbooks and roles

Creating versions of software plays a crucial role in software development projects, including Ansible. It serves as a vital mechanism for monitoring code changes, maintaining historical records, and establishing distinct versions or branches for various project needs. Creating versions in Ansible is important for effective collaboration and project management.

To create versions of your playbooks and roles, use a version control system (VCS) such as Git, which is a widely used tool for managing code repositories. With Git, you can create snapshots (commits) of your code at any point in time, and you can switch between different versions or branches of your code (checkout).

Using Git, you can push your code to a remote repository, such as GitHub or Bitbucket, where you can store it safely and share it with others. You can also pull code from a remote repository to update your local copy with the latest changes.

To effectively use Git with Ansible, follow these steps:

1. Initialize a Git repository in your project directory that houses playbooks and roles.
2. Incorporate a `.gitignore` file to exclude irrelevant files from having versions, such as temporary files or sensitive data.
3. Add and commit your playbooks and roles to the repository, and use descriptive messages for clarity about changes.
4. Create branches for distinct features or environments, such as development, testing, or production.
5. Merge branches when you are ready to integrate changes into the primary branch, which is often labeled `master`.
6. Push your code to a remote repository to help ensure access and collaboration from anywhere.
7. Pull code from the remote repository to sync your local copy with the latest updates.

### 2.7.2 Common scenarios when using Git with Ansible

How you start using Git depends on whether you create a source code repository from scratch, contribute to an existing repository, or create a fork or branch of an existing repository to create an alternative repository.



## Creating a Git repository from scratch

If you have a collection of Ansible playbooks and associated files that you want to start managing by using Git version control, create a repository by running the `git init` command. This command creates a local Git repository, which you can then push (run `push`) to a remote repository to enable collaboration and access to remotely stored copies of your code. Sample commands for creating a repository with `git init` are shown in Example 2-39. It is a best practice that each repository has a `README.md` file that describes the repository.

*Example 2-39 Creating a local repository by running `git init`*

---

```
% ls
ansible.cfg inventory update-hosts.yaml
% echo "# My first repo" > README.md
% git init
Initialized empty Git repository in /<pathname>/.git/
% git add README.md ansible.cfg inventory update-hosts.yaml
% git commit -m "first commit"
[main (root-commit) b7bb240] first commit
 4 files changed, 289 insertions(+)
 create mode 100644 README.md
 create mode 100644 ansible.cfg
 create mode 100644 inventory
 create mode 100644 update-hosts.yaml
% git branch -M main
```

---

To push the local repository to GitHub, first create a repository at [GitHub](#). Your repository can be either private or public. The commands in Example 2-40 show how you push your local repository to a remote GitHub that you created and named `my-first-ansible-repo` as the GitHub user username.

*Example 2-40 Pushing a local repository to a remote location*

---

```
% git remote add origin https://github.com/username/my-first-ansible-repo.git
% git push -u origin main
Username for 'https://github.com': username
Password for 'https://username@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 10 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 1.55 KiB | 1.55 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/username/my-first-ansible-repo.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

---

The password for the `git push` command should be a user token that you created at [GitHub](#).

Now that you have your code stored in both local and remote Git repositories, any changes to existing files or newly created files can be added to the repositories with the `git status`, `git add`, `git commit`, and `git push` commands.

A sample session is shown in Example 2-41, where a new playbook that is named `install-pkg.yml` and the updated playbook `update-hosts.yml` are committed to the repository and pushed to the previously created remote repository.

*Example 2-41 Commit changes to a remote repository*

---

```
% git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   update-hosts.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    install-pkg.yml

no changes added to commit (use "git add" and/or "git commit -a")

% git add update-hosts.yml install-pkg.yml
% git commit -m "my second commit"
[main 50ce021] my second commit
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 install-pkg.yml
% git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
% git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 10 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 345 bytes | 345.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/username/my-first-ansible-repo.git
   b7bb240..50ce021  main -> main
```

---

## Contributing to an existing GitHub repository

You might be part of a team that has a Git repository where multiple team members contribute code to the repository. If so, you start by first cloning the repository to your local machine. Example 2-42 shows how you clone a repository that is called ‘my-team-repo’ to your local machine.

*Example 2-42 Cloning a remote repository with git clone*

---

```
$ git clone https://github.com/username/my-team-repo.git
Cloning into 'my-team-repo'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (6/6), done.
```

```
remote: Total 10 (delta 2), reused 10 (delta 2), pack-reused 0
Receiving objects: 100% (10/10), done.
Resolving deltas: 100% (2/2), done.
$ cd my-team-repo/
$ ls
ansible.cfg  install-pkg.yaml  inventory  README.md  update-hosts.yaml
```

---

Modifications and additions to the local copy of the repository can be committed to the remote repository by using the methods that are shown in Example 2-41 on page 98. It is a best practice to regularly run `git pull`, as shown in Example 2-43, on your local repository to pull other contributors' changes.

*Example 2-43 Using git pull to keep a repository up to date*

---

```
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), 269 bytes | 134.00 KiB/s, done.
From https://github.com/username/my-team-repo
   50ce021..1d4978e  main       -> origin/main
Updating 50ce021..1d4978e
Fast-forward
 inventory | 1 +
 1 file changed, 1 insertion(+)
```

---

## Creating a fork or branch of an existing repository

Forking a repository creates a copy of that repository under your account so that you can modify code from the original repository without affecting it. For example, you might want to fork your own version of a repository like `ansible-power-aix` to experiment with code changes that are specific to your environment.

You can use branches of a repository to support different versions of an application or create features.

Branches can later be merged with the original repository, if appropriate, by running `git merge`, which is usually initiated by a pull request that is submitted to the original repository administrator. The pull request informs the repository administrator that there are committed changes to a branch that you want to merge with the original repository. The administrator might merge the committed changes of the branch to the original repository.

**Note:** For comprehensive guidance about using Git with Ansible, see [Ansible Best Practices - Content Organization](#).

## Documenting playbooks and roles

Documentation is a pivotal means of describing code purpose, functions, and usage instructions. This practice holds significance across software development, and Ansible is no exception.

To adeptly document your Ansible playbooks and roles, comments and metadata files serve as vital tools. Comments, although they are ignored by Ansible, provide human-readable context. Metadata files, which are structured as YAML files, house key-value pairs that elucidate code characteristics.

For effective documentation by using comments and metadata files, consider the following actions:

- ▶ Use comments within playbooks and roles to explain tasks, handlers, variables, or templates, in addition to assumptions and dependencies.
- ▶ Employ metadata files in roles to provide information, such as role name, description, author, license, platforms, dependencies, tags, variables, examples, and more.
- ▶ Use the `ansible-doc` command to generate documentation from metadata files in HTML or plain text format.
- ▶ Use the `ansible-galaxy` command to upload roles to Ansible Galaxy, which is a public repository of roles that anyone can download.

**Note:** For more information and detailed guidance, see the following resources:

- ▶ [Ansible Best Practices - Task and Handler Organization for a Role](#)
- ▶ [Ansible Developing Modules - Documenting Modules](#)
- ▶ [Ansible Galaxy - Creating a Role](#)

## 2.8 Testing and validating playbooks and roles

Testing and validating playbooks and roles is an important part of Ansible automation. These tasks help ensure that your playbooks and roles work as expected and avoid errors or failures when deploying them to your target hosts. This section covers some basic guidelines for writing, testing, and validating your playbooks and roles by using Ansible tools and best practices.

### 2.8.1 Testing playbooks and roles

Testing your playbooks and roles is essential to help ensure that they work as intended and do not cause any unexpected or unwanted effects on your target hosts. There are several ways to test your playbooks and roles by using Ansible tools:

- ▶ Check mode
- ▶ Modules
- ▶ Linting
- ▶ Integration testing

#### Check mode

You can use the `--check` flag when running a playbook or a role to see what changes can be made without applying them. This tool can help you spot any errors or inconsistencies in your code before running it. Check mode does not run scripts or commands, so you must disable it for some tasks by using `check_mode: false`.

#### Modules

You can use certain modules that are useful for testing, such as `assert`, `fail`, `debug`, `uri`, `shell`, or `command`. These modules can help you verify the state or output of your target hosts, check for certain conditions or values, display messages or variables, or run arbitrary commands or scripts.

## Linting

You can use `ansible-lint` to check your playbooks and roles for syntax errors, formatting issues, best practices violations, or potential bugs. `ansible-lint` can also be integrated with other tools such as editors, IDEs, continuous integration and continuous deployment (CI/CD) pipelines, or pre-commit hooks.

## Integration testing

You can use integration testing to test your playbooks and roles against different configurations or environments. For example, you can use Vagrant, Docker or cloud VMs to create isolated test hosts with different OSs or versions. You can also use inventory files or variables to define different parameters for your test hosts, such as package versions or service states.

## 2.8.2 Validating playbooks and roles

Validating your playbooks and roles is the final step before deploying them to your production hosts. It helps ensure that your playbooks and roles perform the wanted actions and produce the expected results on your target hosts. There are several ways to validate your playbooks and roles by using Ansible tools:

- ▶ Dry run
- ▶ Idempotence
- ▶ Verbose
- ▶ Notifications
- ▶ Reports

### Dry run

You can use the `--diff` flag when running a playbook or a role in check mode to see the differences between the current state and the wanted state of your target hosts. This tool can help you verify that the changes are correct and complete before applying them.

### Idempotence

You can run your playbook or role multiple times on the same target host to check that it is idempotent, which means that it cannot make any changes on subsequent runs unless the state of the host changed externally. Idempotence helps ensure that your playbook or role is consistent and reliable.

### Verbose

With the verbose option, you can see more details about what Ansible is doing when it runs your playbooks and roles. You can use different levels of verbosity, from `-v` to `-vvvv` to increase the amount of information that is displayed. The verbose option can help you debug your Ansible code, identify errors, and check the results of your tasks.

### Notifications

You can use handlers to trigger notifications when certain tasks make changes on your target hosts. For example, you can use handlers to restart a service, reload a configuration file, or send an email alert. Handlers can help you validate that your changes have taken effect on your target hosts.

## Reports

You can use callbacks or plug-ins to generate reports on the running of your playbooks or roles. For example, you can use callbacks to display statistics, summaries, logs, or graphs of your playbook or role runs. You can also use plug-ins to send reports to external systems or services, such as Slack, email, or webhooks. Reports can help you validate that your playbooks or roles ran successfully and without errors.

**Note:** For more information and detailed insights about testing and validating playbooks and roles, see the following sources:

- ▶ [Five Questions for Testing Ansible Playbooks and Roles](#)
- ▶ [Ansible Testing Strategies](#)
- ▶ [Introduction to Ansible Playbooks](#)



## Getting started with Ansible

So far, you have learned about the advantages of using automation and specifically have seen that Ansible can be an excellent choice for automation because it is a single solution that helps automate many environments, which include networks, cloud resources, and servers. This chapter shows how to get started on your automation journey with Ansible in your IBM Power environments.

This chapter describes the different architectures that you might want to consider when designing your Ansible automation environment, depending on your business requirements. It also provides guidance about choosing the right server (or it might be more than one server) for the Ansible Controller, and then shows how to install Ansible Controller in your IBM Power environment and configure your different IBM Power based logical partitions (LPARs) as Ansible clients.

The following topics are described in this chapter:

- ▶ Designing your Ansible environment
- ▶ Choosing the Ansible Controller node
- ▶ Installing your Ansible control node
- ▶ Preparing your systems to be Ansible clients

## 3.1 Designing your Ansible environment

There are several stages of Ansible adoption. Depending on your Ansible experience, your environment size, and your automation expectations, you might want to start small at the beginning and then grow through different stages as you expand your automation platform. However, you might also choose to go directly to designing and developing a fully scalable Ansible environment that you use to support hundreds of developers and users. This section helps you understand the different architectures that you can choose for your Ansible environment based on your business requirements.

### 3.1.1 Starting simple: Ansible Core and Ansible Community

If you have never worked with Ansible or are starting your automation journey with IBM Power and Ansible, do it as simply as possible. Do not over complicate things, and keep it simple. The most important outcome on this stage is to get your first tasks automated and provide a positive experience for your users.

In this simple case, the architecture for your Ansible installation might look like Figure 3-1.

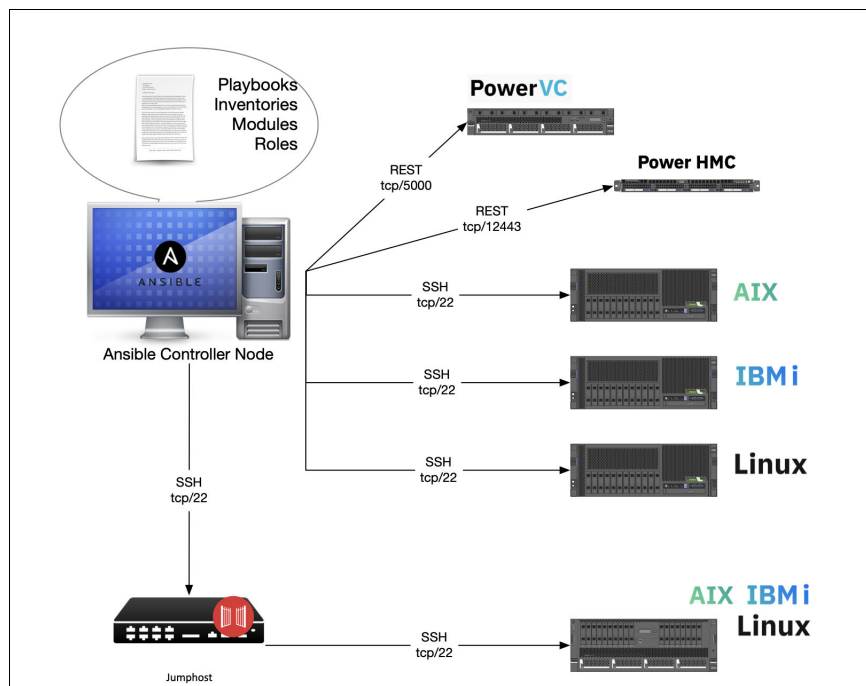


Figure 3-1 Using Ansible Core or Ansible Community to manage your IBM Power environment

Your first step is to choose the right server for your Ansible Controller node. The Ansible Controller node is a server where you install Ansible and develop your playbooks. There are only two requirements for this controller node server:

1. You can install Ansible on it.
2. The server has Secure Shell (SSH) access to the IBM Power servers and other devices that you want to manage.

Satisfying the first requirement is simple because Ansible can be installed on any supported operating system (OS) that runs on IBM Power. For more information about choosing an OS for Ansible Controller node, see 3.2, “Choosing the Ansible Controller node” on page 121.



Ansible primarily works by using SSH connections to the managed devices, although there are some exceptions. For example, in an IBM Power environment, for both IBM PowerVC and the IBM Power Hardware Management Console (HMC), Ansible works by using the Representational State Transfer (REST) application programming interfaces (APIs) that are provided by those products.

If you plan to automate your Linux, AIX, or IBM i LPARs, you need SSH access to them. Ansible supports the usage of SSH gateways (jump servers or bastion hosts) for access to devices with extra security requirements.

## Managing your automation playbooks

Before you automate your first task, you must make one more decision: Where will you save your work? If you are the only playbook developer, your home directory is the perfect place. If you have two or more people who work with playbooks, create a directory that can be accessed by all the developers.

Consider using a source control system like Git to track the history of changes in the playbook. Although it is not necessary, it is always a best practice and makes development transparent and traceable.

After you choose your Ansible Controller node and decide where to save your future work, install Ansible by following the instructions in 3.3, “Installing your Ansible control node” on page 121.

## Get started

After you install Ansible on your controller node, automate your first task. Consider the following factors when you choose this first task:

- ▶ Choose a simple task that you do regularly.
- ▶ Keep it simple. Avoid perfectionism.
- ▶ Automate step by step. Do not try to develop the whole playbook at once.
- ▶ Measure the outcome. How much time did you save?
- ▶ Speak to your colleagues about your success. Spread the word about Ansible.

**Tip:** In this simple architecture, use either Ansible Core or Ansible Community for your Ansible Controller. Both of these products are supported by the Ansible community and not supported by any vendor. Consider whether you need a better and more reliable support option. As you move into more production environments, consider using a vendor-supported product such as Ansible Automation Platform, which supports Ansible. For more information about Ansible Automation Platform, see 3.1.2, “Scaling up: Ansible Automation Platform” on page 105.

### 3.1.2 Scaling up: Ansible Automation Platform

When you are starting out and designing the environment for one or two teams, you might need to use only one of the components of Ansible Automation Platform, that is, the Ansible Automation Controller (formerly known as Ansible Tower). Ansible Automation Controller can be installed on IBM Power or on any x86 server.

If you have experience with base Ansible Core and have several team members, you might want to use Ansible Automation Platform, especially if you have several administrators who develop playbooks and many users who run only the playbooks.

Ansible Automation Platform provides role-based access control (RBAC) for your Ansible environment to define the rights of your automation users, that is, which playbooks they can run and what parameters they may use in those playbooks. It also defines the rights of automation developers by defining which projects they are working on.

The GUI of Ansible Automation Platform removes the complexity of running single commands and remembering the appropriate parameters to use in an Ansible playbook.

Another feature of Ansible Automation Platform is support for execution environments. Execution environments are container images that make it possible to incorporate system-level dependencies and collection-based content. Each execution environment can have a customized image to run jobs, and each of them contain only what you need when running the job. Using the execution environments, you can predefine Ansible environments for automation users so that they no longer have problems managing the different parameters that are presented by multiple versions of modules, roles, and collections. You effectively define the “single source of truth” for your automation.

Ansible Automation Platform is fully supported by Red Hat. If you are designing an automation platform for an enterprise, support for Ansible is one of the most important considerations, and you must clarify your requirements before starting. Ansible Core and Ansible Community are open-source projects that come with only community support through the projects code repositories on GitHub. What might be the impact if your enterprise automation solution has an issue and you depend on the community members, many of whom do not understand IBM Power to find a solution to your problems? Therefore, it makes sense to have a support contract with Red Hat for any enterprise automation environment.

At the time of writing, Red Hat supports Ansible Automation Platform on IBM Power running Red Hat Enterprise Linux (RHEL) as a “technology preview”. Red Hat cannot guarantee the stability of all features on IBM Power, but attempts to resolve any issues that customers experience. For more information about technology preview support, see this [Red Hat document](#).

At the time of writing, Red Hat has not stated when Ansible Automation Platform on IBM Power will be fully supported, but you can expect it soon. When running Ansible Automation Platform on RHEL, you can manage systems that run any Linux distribution, AIX, or IBM i as supported use cases, but Ansible Automation Platform can be installed only on RHEL.

## **Ansible Automation Controller requirements**

To run Ansible Automation Controller, you need RHEL 8.6 or later and at least 16 GB of memory. The memory requirement for the automation controller depends on the maximum number of hosts that the system is expected to configure in parallel. This number is managed by the forks configuration parameter in the job template or system configuration. You must increase memory as you expand the number of forks to support more systems. Red Hat recommends 1 GB of memory per 10 forks, and 2 GB for the automation controller services. If you set the forks parameter to 400, then the automation controller requires 42 GB of memory.

You need at least 40 GB of available space in /var on your server to proceed with the installation. You might need even more if you have many playbooks and different execution environments. To get a suitable performance for your installation, the storage should be capable of a minimum of 1500 IOPS.

When using Automation Controller, you can store all your playbooks locally on the server where the Automation Controller is installed by creating one or more subdirectories under `/var/lib/awx/projects/` and copying your playbooks there. This task is simple if you have one automation developer, but as you add more automation developers all working on different projects, you might need several different directories, and you must manage file permissions for all users and projects. A flat file system containing files does not have any source control mechanism that is built in, so as a best practice, use GitHub, GitLab, or similar software to store your playbooks source code centrally and enable collaboration between automation developers.

Figure 3-2 shows the usage of Ansible Automation Controller managing an IBM Power infrastructure.

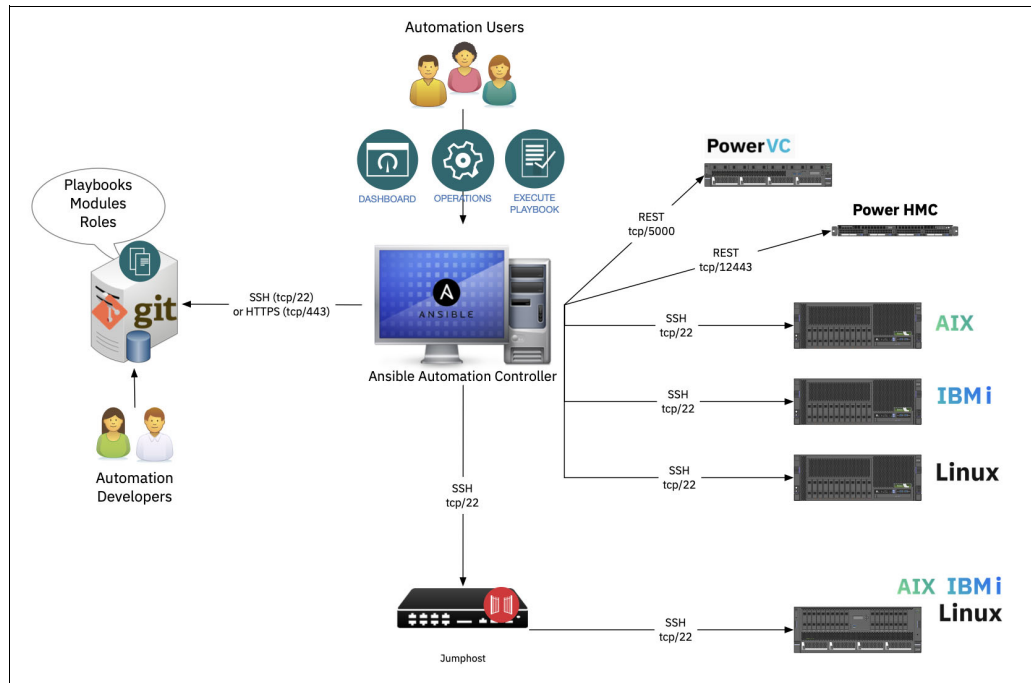


Figure 3-2 Using Ansible Automation Controller to manage an IBM Power infrastructure

As you can see in Figure 3-2, the architecture is almost the same as the one that uses Ansible Core. However, it supports more developers and users of your automation projects, and it provides Red Hat support.

## Reference architectures

This section provides some sample Ansible architectures with different combinations of Ansible automation components to meet different availability requirements for your department or business. These Ansible automation components can be deployed on a single system in some environments or might require more than one system as you build out your Ansible Automation Controller or Platform.

### Reference architecture 1: Ansible Automation Controller

This architecture requires minimum computing resources to build an Ansible Automation Controller environment. The architecture is shown in Figure 3-3.

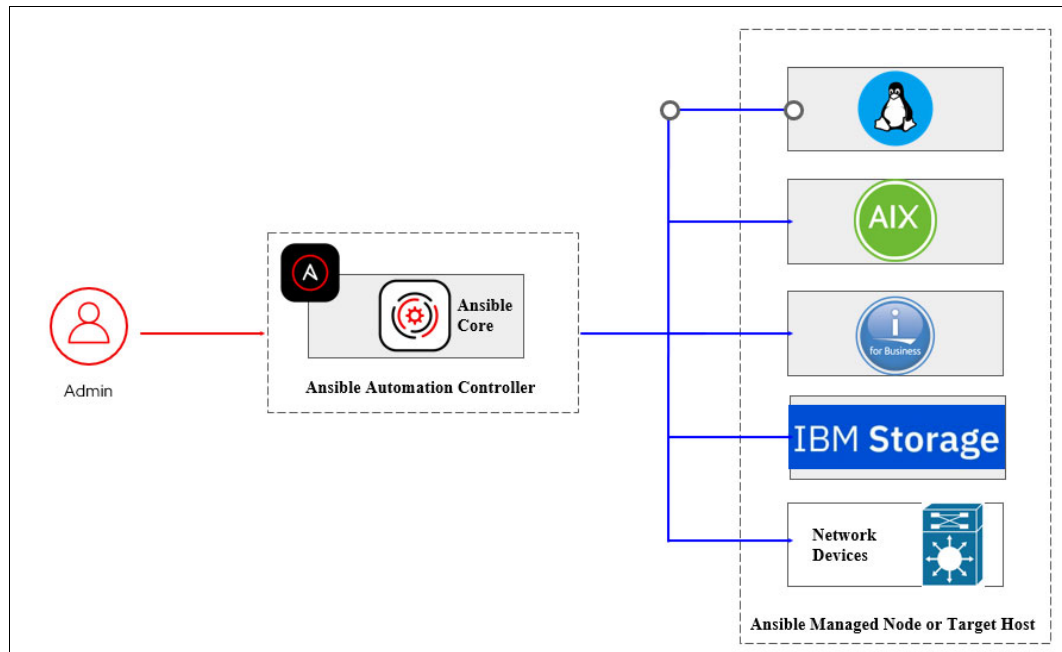


Figure 3-3 Reference architecture 1: Ansible Automation Controller

This architecture is defined by the following items:

- ▶ Deploy Ansible core in a single system.
- ▶ Number of nodes that are required: One Ansible automation controller node.
- ▶ Automation: Any.
- ▶ Use cases: Testing and development environment.

CLI-driven, audit logs, and access controls depend on third-party software integration, no high availability (HA), and complicated Day 2 operation. No additional subscription or license is required. Nearly any UNIX like machine with Python 3.9 or later is supported as the installation target, which includes RHEL, IBM AIX, Debian, Ubuntu, macOS, BSDs, and Windows under a Windows Subsystem for Linux (WSL) distribution.

This architecture is a good choice for a small starter system for test and development or for a small environment with non-critical automation requirements. Support is community-based.

## Reference architecture 2: Automation Platform - All-in-one without Automation Hub

This architecture requires minimum computing resources to build an Ansible Automation Platform environment. The architecture is shown in Figure 3-4.

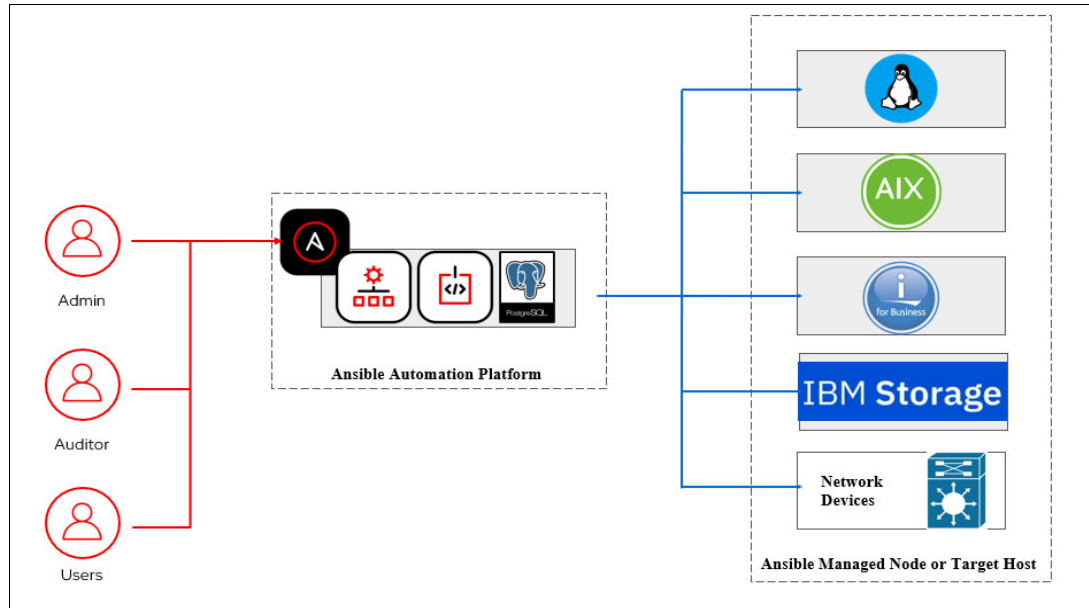


Figure 3-4 Reference architecture 2: All-in-one without Automation Hub

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform in a single system.
- ▶ Number of nodes that are required: One all-in-one automation controller node.
- ▶ Automation: Any.
- ▶ Use cases: Testing and development environment.

Provides a GUI, audit logs, and an access control built-in feature. Also provides a Day 2 operation automation platform. Subscriptions or licenses are required. This version does not provide HA and does not use Automation Hub to manage execution images or Ansible Content Collections management. The supported deployment environment is limited to RHEL OSs on x86 or Power servers.

This architecture is a good choice for a small starter system for test and development, or for a small environment with non-critical automation requirements that also provides a base for expansion as you grow. Support is provided by Red Hat.

### Reference architecture 3: All-in-one with Automation Hub

This architecture builds on “Reference architecture 2: Automation Platform - All-in-one without Automation Hub” on page 109 by adding the Automation Hub, as shown in Figure 3-5.

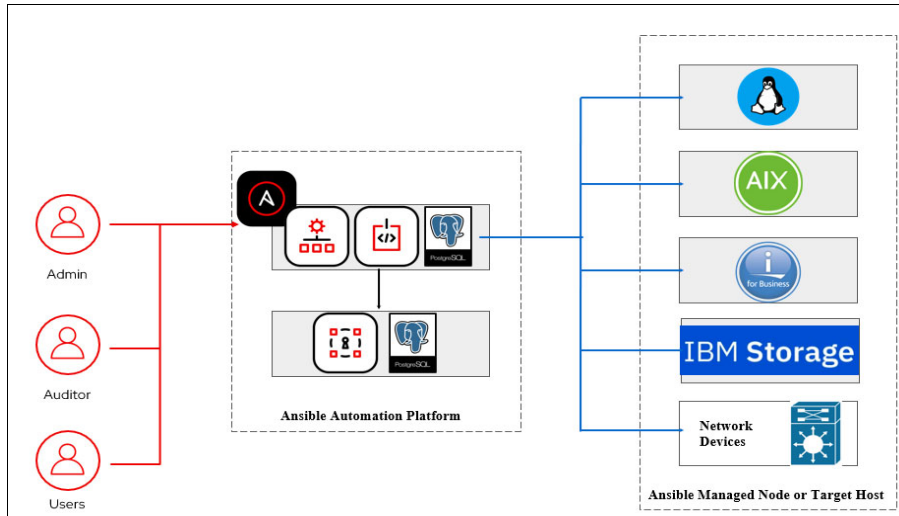


Figure 3-5 Reference architecture 3: All-in-one with Automation Hub

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform in a single system and Automation Hub.
- ▶ Number of nodes that are required:
  - One all-in-one automation controller node
  - One all-in-one Automation Hub node
- ▶ Automation: Any.
- ▶ Use cases: Testing and development environment.

This architecture is an enhancement of “Reference architecture 2: Automation Platform - All-in-one without Automation Hub” on page 109 by adding an All-In-One Automation Hub. The Automation Hub manages your Ansible playbooks and collections, which can help you automate new projects faster and more reliably. There is still no HA, and service redundancy is not considered.

This architecture is a good choice for a small starter system for test and development or for a small production environment with non-critical automation requirements. This architecture has the basics of a larger, scaled-up solution to support growth in your environment. Support is provided by Red Hat.

### Reference architecture 4: External database server without high availability

This architecture uses an external database provider as the database system, as shown in Figure 3-6.

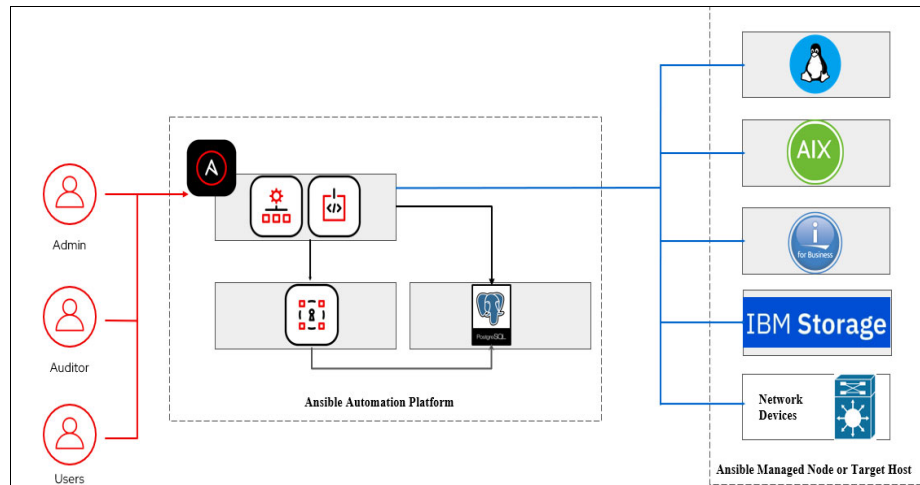


Figure 3-6 Reference architecture 4: External database server without high availability

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform in a single system and Automation Hub. Use an external database provider as the database system.
- ▶ Number of nodes that are required:
  - One automation controller node
  - One Automation Hub node
  - One shared database node
- ▶ Automation: Any.
- ▶ Use cases: Testing and development environment.

This architecture is an enhancement of “Reference architecture 3: All-in-one with Automation Hub” on page 110 by adding a separate external databases system and separate system access control. There is no HA or service redundancy.

This architecture is a good choice for a medium to large system for test and development or for a medium-sized environment with non-critical automation requirements. Support is provided by Red Hat. This environment scales well in terms of the number of systems that are managed and the number of projects that are managed, but it does not provide any way to recover the automation platform if the controller nodes fail.

### Reference architecture 5: External database server with high availability except for the database server

This architecture builds on “Reference architecture 4: External database server without high availability” on page 111 by adding HA for the Ansible Controller portions, as shown in Figure 3-7.

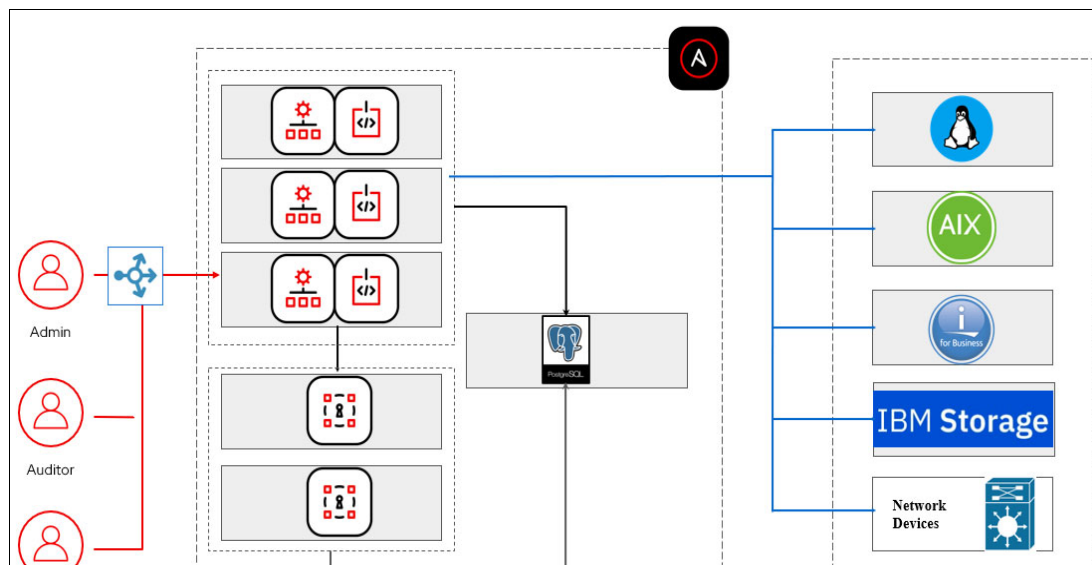


Figure 3-7 Reference architecture 5: External database server with high availability except for the database server

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform and Automation Hub in multiple systems. Use an external database provider as the database system without HA.
- ▶ Number of nodes that are required:
  - Three automation controller nodes
  - Two Automation Hub nodes
  - One shared database node
- ▶ Automation: Any.
- ▶ Use cases: Production environment.

This architecture is an enhancement of “Reference architecture 4: External database server without high availability” on page 111 by adding HA except for database server HA. The database server can recover from backup if needed, but the RPO and RTO will be higher.

This architecture is aimed at medium to large environments that have critical requirements for their automation environment because there are redundant nodes for the automation controller and Automation Hub functions.



### Reference architecture 6: External database server with full high availability

This architecture adds HA to “Reference architecture 5: External database server with high availability except for the database server” on page 112, as shown in Figure 3-8.

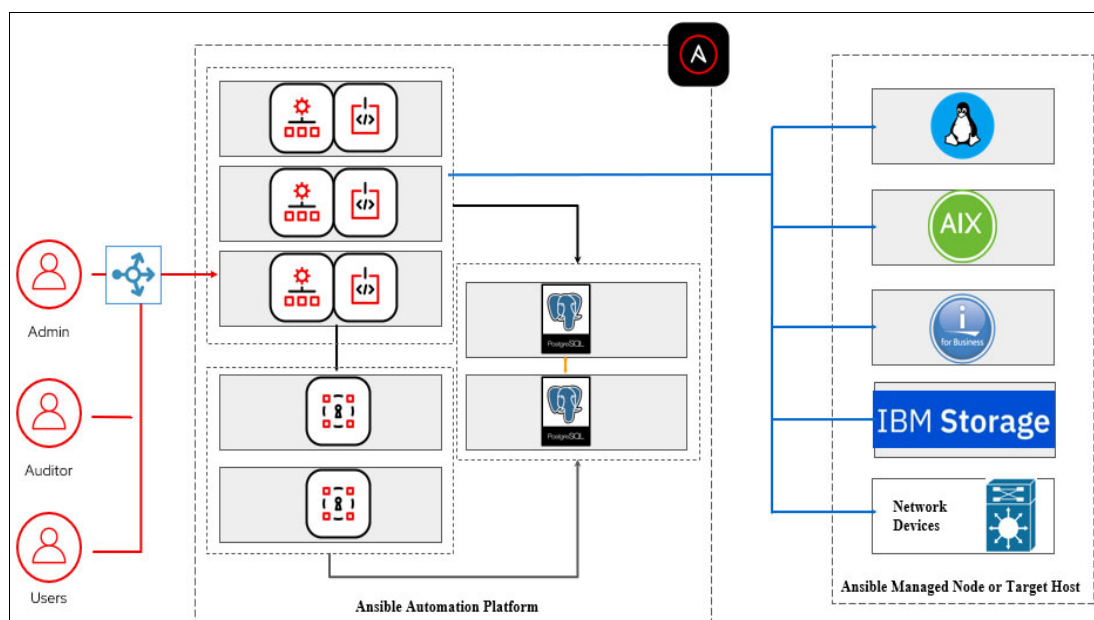


Figure 3-8 Reference architecture 6: External database server with full high availability

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform and Automation Hub in multiple systems. Use an external database provider as the database system with HA.
- ▶ Number of nodes that are required:
  - Three automation controller nodes
  - Two Automation Hub nodes
  - Two shared database nodes
- ▶ Automation: Any.
- ▶ Use cases: Production environment.

This architecture is an enhancement of “Reference architecture 5: External database server with high availability except for the database server” on page 112 by adding HA for all components. The database server can recover from backup if needed. Reduces the RPO and RTO. Automation Platform is considered the important service, but enabling access between automation controller nodes and managed nodes within the same data center with different network zones or segmentation might be complicated.

This architecture is a great option for a highly available and highly scalable automation environment within a single site. It lacks only the ability for quick recovery to a second site in a site failure.

### Reference architecture 7: External database server with full high availability and a separate execution node

This architecture adds separate execution zones to “Reference architecture 6: External database server with full high availability” on page 113, as shown in Figure 3-9.

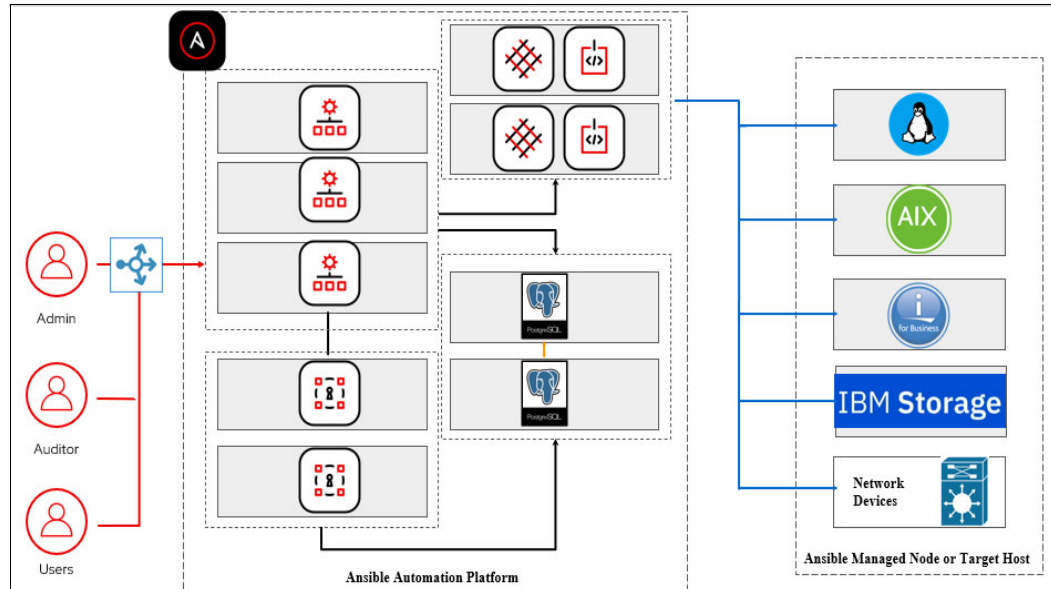


Figure 3-9 Reference architecture 7: External database server with full high availability and a separate execution node

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform and Automation Hub in multiple systems. Use an external database provider as the database system with HA. Deploy separate execution environments in different network zones.
- ▶ Number of nodes that are required:
  - Three automation controller nodes
  - Two execution nodes
  - Two Automation Hub nodes
  - Two shared database nodes
- ▶ Automation: Any.
- ▶ Use cases: Production environment.

This architecture is an enhancement of “Reference architecture 6: External database server with full high availability” on page 113 by adding separate execution environments in different network zones. This architecture minimizes firewall rules changes to enable access between automation controller nodes and managed nodes within the same data center with different network zones or segmentation. Site resilience is still missing.

This architecture provides a fully scalable and highly available automation solution and includes more separation of automation activities for security. It includes redundant components, but still does not address disaster recovery (DR).

**Reference architecture 8: External database server with full high availability and disaster recovery - independent operation**

This architecture adds a DR environment. Automation can be managed from either the primary location or the DR location, as shown in Figure 3-10.

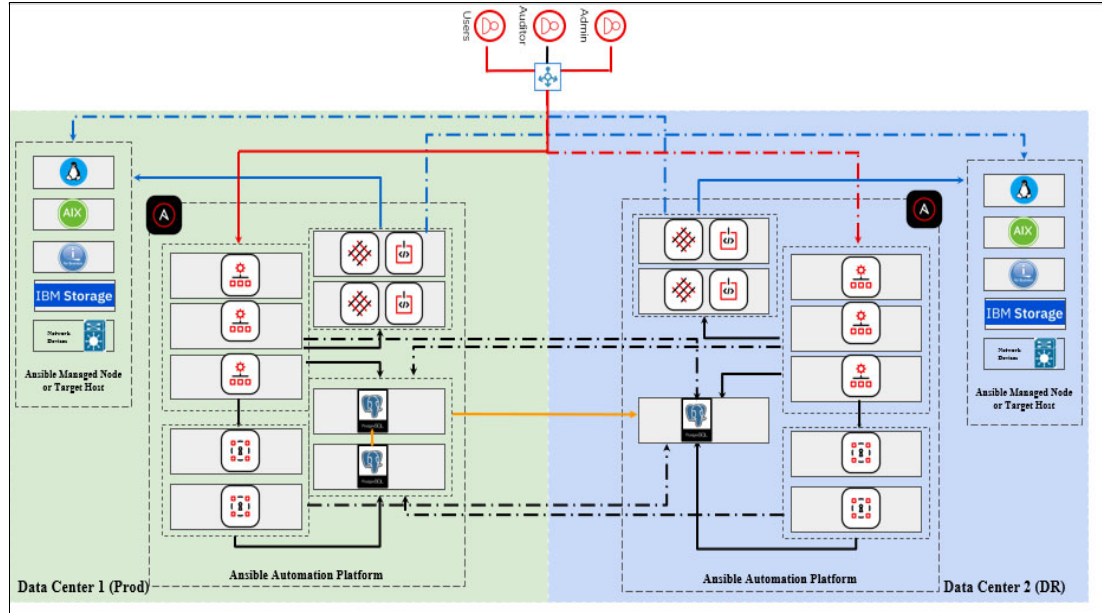


Figure 3-10 Reference architecture 8: External database server with full high availability and disaster recovery - independent operation

This architecture is defined by the following items:

- ▶ Deploy Ansible Automation Platform and Automation Hub in multiple systems. Use an external database provider as the database system with HA. Deploy separate execution environments in different network zones.
- ▶ Number of nodes that are required:
  - Three automation controller nodes per site
  - Two execution nodes per site
  - Two Automation Hub nodes per site
  - Two shared database nodes in the production site
  - One shared database node in the DR site
- ▶ Automation: Any.
- ▶ Use cases:
  - Production environment
  - DR environment

This architecture is an enhancement of “Reference architecture 7: External database server with full high availability and a separate execution node” on page 114 by adding a separate DR environment. At least one automation platform from either site can perform the automation operation on all managed nodes across the site. The automation operation on all the managed nodes across the site is not impacted if one of the automation platforms is down. Significant firewall rules changes might be required to enable access among automation controller nodes and managed nodes across the sites. This architecture is a fully scalable and redundant solution with DR.

**Reference architecture 9: External database server with full high availability and disaster recovery - joint operation**

This architecture adds a joint operation to “Reference architecture 8: External database server with full high availability and disaster recovery - independent operation” on page 115, as shown in Figure 3-11.

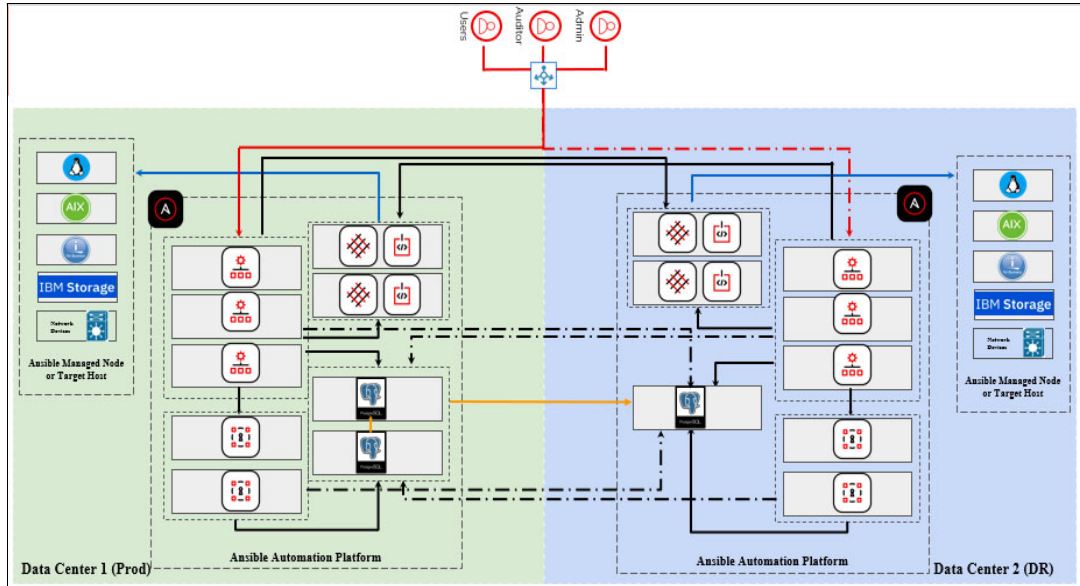


Figure 3-11 Reference architecture 9: External database server with full high availability and disaster recovery - joint operation

This architecture is defined by the following items:

- ▶ Deploy Anible Automation Platform and Automation Hub in multiple systems. Use an external database provider as the database system with HA. Deploy separate execution environments in different network zones.
- ▶ Number of nodes that are required:
  - Three automation controller nodes per site
  - Two execution nodes per site
  - Two Automation Hub nodes per site
  - Two shared database nodes in the production site
  - One shared database node in the DR site
- ▶ Automation: Any.
- ▶ Use cases:
  - Production environment
  - DR environment

This architecture is an enhancement of “Reference architecture 8: External database server with full high availability and disaster recovery - independent operation” on page 115 by adding a separate DR environment. One Automation Platform from either site can perform the automation operation on all managed nodes across the site through the execution node in the respective site. The automation operation on all managed nodes across the site is not impacted if one of the automation platforms is partially down, except for the execution nodes.

There are minimal firewall rules changes to enable access among automation controller nodes and managed nodes across the site. Firewall rules changes are required only for the access between automation controller nodes and execution nodes across the site only. Other required access is maintained within the site only.

Table 3-1 helps differentiate the implementation considerations between “Reference architecture 8: External database server with full high availability and disaster recovery - independent operation” on page 115 and “Reference architecture 9: External database server with full high availability and disaster recovery - joint operation” on page 116.

*Table 3-1 Differentiation between reference architecture 8 and reference architecture 9*

<b>Characteristic</b>	<b>Reference architecture 8</b>	<b>Reference architecture 9</b>
Operation dependency	Independent and tolerant of any full site failures.	Partially independent and tolerant of any partial site failures. The execution nodes must be accessible for automation activities to work across the sites.
Execution node	The control plane (nodes) is connected to execution nodes on the same site only.	The control plane (nodes) is connected to all execution nodes on both sites.
Firewall rules changes	In a large automation environment, many firewall rules must be changed.	In a large automation environment, a minimum number of firewall rules must be changed.
Network bandwidth utilization	In a large automation environment, the network bandwidth utilization is comparatively high across the sites.	In a large automation environment, the network bandwidth utilization is minimized across the sites.
Network latency	Not suitable for high network latency across the sites. The network latency between the sites must be negligible so that the execution nodes with automation execution environments and the target host or endpoints can be placed in two different locations to enable automation for edge use cases.	Suitable for high network latency across the sites. Because the execution plane (nodes) runs only user-space jobs, they may be geographically separated, with high latency, from the control plane (nodes). The execution nodes with automation execution environments are placed in different locations that are closer to the target host or endpoints to reduce latency and enable automation for edge use cases.

**Conclusion**

These reference architectures provide guidance about how to design your automation environment and create your supported architecture by choosing the components that meet your requirements best. However, there are further things to consider for when adding and integrating additional solution components.

Consider the following points as you design your environment:

▶ Database nodes

Database HA clusters can be configured by using an RHEL native HA cluster solution that is called Pacemaker, or you can use the PostgreSQL HA solutions that known as the Primary-Standby and Primary-Primary architectures.

▶ Automation controller nodes

At least two nodes can be considered for automation controller nodes HA.

▶ Automation Hub nodes

Automation Hub is optional, and alternative solutions can be used. You can use one Automation Hub in the DR site if there are any resource limitations.

▶ Third-party services integration

Integrate and configure third-party services as needed, for example:

- Source code management (SCM): Manage project and playbooks through an SCM system, such as Git, Subversion, and Mercuria.
- Notification methods: Use email, Grafana, Slack, or similar tools.
- Authentication: Use LDAP, SAML, or token-based authentication.
- Logging: Consider using logging aggregation services for monitoring and data analysis of your systems, such as Splunk, Loggly, Sumologic, or Elasticstack (formerly ELK stack).

### 3.1.3 Enterprise-ready environment

A real enterprise consists of more than of one team. It has many DevOps teams and environments. It also has many different non-functional requirements for automation, such as the requirement for centrally managed authentication through Active Directory or requirements for delivering security-related logs to the organization's Security Information Event Monitoring (SIEM) system. Many of these requirements are already integrated in Ansible Automation Platform and can be implemented by using those features. However, some of them require third-party software.

In an enterprise environment, all source code must be saved in a source control repository to track changes to the code and see who did what. The repository also enables you to separate projects and teams. The same concepts apply to automation source code, that is, your playbooks and roles. Your Windows administration team has nothing to do with AIX or IBM i automation. AIX operations usually do not interfere with Microsoft SQL Server or Sharepoint resources that are managed by other teams.

Some common control management tools that are used in enterprises are GitHub Enterprise and Gitlab Enterprise. They are based on the open source Git project, which you can use on your Linux, AIX, or IBM i server.

After a change to a source code is committed, the new code must be tested. If someone made a small mistake in the automation code, it can cause problems across your whole application deployment or infrastructure. Testing can be as simple as doing a syntax check or can involve more complex integration testing where the whole infrastructure is built and the application is deployed into a special testing environment. Source control management tools like GitHub Enterprise and Gitlab Enterprise have their own set of continuous integration (CI) tools, but you may also use the open-source tool Jenkins to build your integration pipeline.

Check that the source repository does not contain any passwords, tokens, or other secrets. Your secrets must be stored in Ansible Automation Platform or in another vault tool like Hashicorp Vault, but not in the source code. Modern SCM tools like GitHub and Gitlab can integrate with all common security tools to automate source code scanning.

When the whole testing process completes, you can deploy the code into your production infrastructure. This task may be done automatically by using continuous deployment (CD). When you use Ansible Automation Platform, you receive a new version of a project after synchronizing it. You might want to automate Ansible Automation Platform as you automate your other applications.

In an enterprise environment, you want your automation to be predictable, which is possible if your code is tested before it goes to production. Ensure that the code that you rely on is stable, which includes every role, every module, and every collection. To meet this goal, use the curated Red Hat Automation Hub instead of Ansible Galaxy as the source of your collections. Another option is to use Private Automation Hub, which is provided by Ansible Automation Platform. With Private Automation Hub, you can upload (or synchronize) only the content that you need for your automation code, which enables you or your IT security team to validate and approve the components that are used in your organization.

In the simplest form of deployment, you can use one Ansible Controller to manage all nodes. In an enterprise-grade deployment, you install Ansible Automation Controllers according to your infrastructure requirements. You may have separate controllers for each stage (development, test, and production) or you may install them based on your network configuration (separate controllers for an office network, “normal” servers, high priority servers, or DMZ servers). These extra controllers make your architecture more complex, but it is simpler to control which projects access which resources.

Another component of Ansible Automation Platform that is designed for enterprise environments is event-driven automation (EDA). Consider the following questions:

- ▶ What happens in the environment if someone provisions a server without using your automation?
- ▶ What happens if a new user must be created on several servers?
- ▶ What happens if a file system on a server needs more space?

These scenarios are all use cases for EDA.

With EDA, you develop playbooks for each use case (configure a server, create a user on a server, or expand a file system) and then connect your external systems like PowerVC (provisioning), ticketing system (new user creation), or monitoring (file system) to EDA. Within EDA, you define the policies and rules and EDA runs the appropriate playbooks when a defined event happens so that you can build a fully automated enterprise. For more information about use cases for EDA, see 1.3.4, “Event-driven automation” on page 15.

One more aspect of a complex automation architecture is organizational in nature. Because you have multiple components to manage (several Ansible Automation Controllers, a Private Automation Hub, EDA and others), the job of managing the environment cannot be a side job. In this case, your organization needs a separate automation team that automates EDA, and manages Ansible Automation Platform. This architectural pattern applies only to enterprises with a large infrastructure and many applications.

Figure 3-12 provides an example of this complex environment. Your environment might be even more complex, and you might need to spend time defining and building your automation architecture.

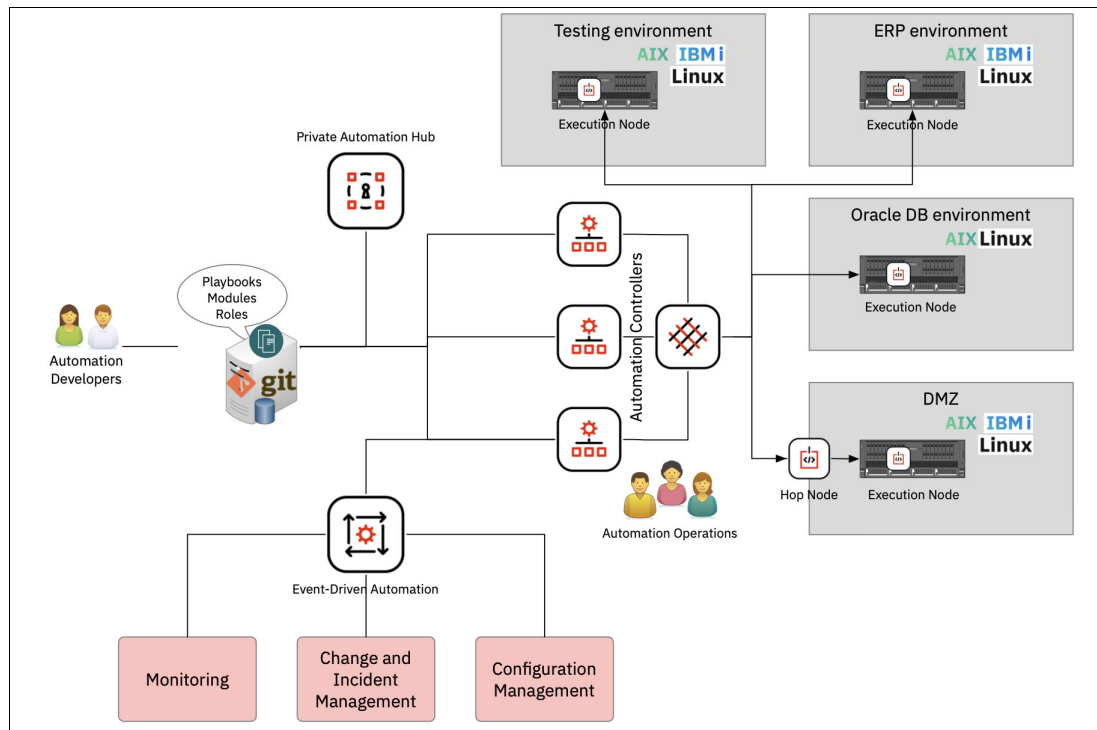


Figure 3-12 Sample Ansible Automation Platform implementation

### 3.1.4 Developing an “automation first” attitude

How you start with Ansible depends on your team, your budget, and your timeline. You could start with the simplest form of deployment and later grow into a full-pledged Ansible Automation Platform installation as you integrate automation across your enterprise (including third-party applications). However, if you are ready to design an automation architecture for your whole enterprise, you can start with Ansible Automation Platform.

Automation and automation practices evolve. Your environment is live, so be prepared to enhance it every time that you require a new feature or a new integration, and be prepared to eliminate unnecessary or unused features.

The most important step in the automation plan is to create an “automation first” environment. Take your time and think: Can I automate this function by using Ansible? The obvious answer is often yes. Then, automate the task and let the job be done by Ansible. When you get to this point, you no longer need root or QSECOFR privileges on your systems. All that you need is that your systems are connected to your automation platform and can run Ansible playbooks there.



## 3.2 Choosing the Ansible Controller node

Ansible can be installed on nearly any system. To choose the best location and system to run your Ansible control node requires that you understand your environment and your automation requirements.

Before choosing the right controller node for Ansible, you must answer a simple question: Which systems do you plan to manage with Ansible? Consider the following cases:

- ▶ If you want to manage only your IBM i database, use your IBM i server as the Ansible Controller.
- ▶ If you want to manage your AIX environment, install Ansible on your Network Installation Manager (NIM) server. NIM is the central point of AIX infrastructure and has access to all AIX servers. Often, NIM already uses OpenSSH connections between the NIM server and the NIM clients.
- ▶ If you have SAP HANA on IBM Power, or other Linux applications on IBM Power, use an existing Linux on Power LPAR to install Ansible. This choice has one significant advantage: Ansible is developed under Linux. At the time of writing, you can install Ansible Core 2.15 on Linux on Power, but Ansible Core 2.14 is the latest version that is available on AIX. Although most modules and collections support Ansible 2.9 or later, if you have something specific that requires a newer Ansible version, your only choice is Linux.
- ▶ You may use Linux on x86 for the Ansible Controller node.

The Ansible Controller node must have SSH access to all systems that you want to manage. Help ensure that the connection can be made through firewalls and security zones.

You might want to have your Ansible Controller node as close to the managed servers as possible. If you place your Ansible Controller node in the DMZ with other servers, it simplifies the connection between the Ansible Controller and the target hosts, but it might also be difficult to upload your playbooks and roles to it. It might be simpler to install Ansible on a server outside of the DMZ and use some jump host for playbook execution.

## 3.3 Installing your Ansible control node

This section describes how to install the Ansible code on your IBM Power controller node. Ansible is an excellent tool for configuration management, automated deployment, and orchestration. There are two components to consider when using Ansible

- ▶ The control node that runs the playbooks and manages the automation.
- ▶ The managed node or client, which is the automated device (often called the target machine or device).

Ansible is agentless, which means that it can communicate with machines or devices without requiring that an application or service is installed on that managed node. It is one of the main differences between Ansible and other similar applications like Puppet, Chef, CFEngine, and Salt.

The Ansible Controller is often called Ansible Engine (old name) or Ansible Core (new name). Ansible Core provides a CLI to manage your Ansible automation environment. For some administrators, the CLI-based approach is intimidating and they are looking for a GUI instead. For GUI-based management, you can choose to use the Ansible Automation Platform, which provides a GUI interface for Ansible Core and more management capabilities.

Ansible Core is available and supported on all OSs that are supported by IBM Power: AIX, IBM i, and Linux on Power. Ansible Automation Platform is also available for IBM Power environments, but it is supported only by Linux on Power. At the time of writing, Linux on Power support for Ansible Automation Platform is in technology preview.

### 3.3.1 Linux as an Ansible Controller

This section describes the following topics:

- ▶ Installing Ansible on RHEL
- ▶ Verifying the installation of Ansible on RHEL
- ▶ Additional preparation and configuration for Ansible on RHEL
- ▶ Ansible Automation Platform installation on RHEL
- ▶ Ansible Automation Controller installation on RHEL

#### Installing Ansible on RHEL

Ansible Engine and Ansible Core cannot be installed simultaneously on an RHEL 8 system. The installer (Red Hat Package Manager) is in two different rpm repositories. Ansible Core is included in RHEL 8.6 and later and RHEL 9 OS under the AppStream repository, and can be installed by running the rpm or dnf (newer version of yum) command. For more information, see [Using Ansible in RHEL 8.6 and later](#).

To install Ansible on RHEL, complete the following steps:

1. Verify the system identity, name, organization name, and organization ID that you received when you registered for a subscription, as shown in Example 3-1.

*Example 3-1 Verifying the subscription-manager details*

---

```
# subscription-manager identity
system identity: 8e73cf0a-4651-4b0d-95c1-b0b73a886785
name: app24allinone.example.com
org name: 11009103
org ID: 11009103
```

---

2. Verify the status of the products and attached subscriptions for the system, as shown in Example 3-2.

*Example 3-2 Verifying the subscription-manager status*

---

```
# subscription-manager status
+-----+
  System Status Details
+-----+
Overall Status: Disabled
Content Access Mode is set to Simple Content Access. This host has access to
content, regardless of subscription status.
System Purpose Status: Disabled
```

---

3. Verify that the Red Hat Package Manager repository is configured and enabled by using the command that is shown in Example 3-3 on page 123.

*Example 3-3 Verifying the Red Hat Package Manager repository configuration*

```
# yum repolist
Updating Subscription Management repositories.
repo id                                repo name
rhel-8-for-ppc64le-appstream-rpms      Red Hat Enterprise Linux 8
for ppc64le - AppStream (RPMs)
rhel-8-for-ppc64le-baseos-rpms         Red Hat Enterprise Linux 8
for ppc64le - BaseOS (RPMs)
```

4. Install the `ansible-core` rpm in the system by running the following command:

```
# dnf install ansible-core python3-virtualenv vim
```

**Note:** Help ensure that your system is connected to the correct rpm repository. If the system directly connects with the internet, then help ensure that the subscription is configured and enable the correct repository. For more information about using subscription manager, see [How to register and subscribe an RHEL system to the Red Hat Customer Portal using Red Hat Subscription-Manager](#).

### Verifying the installation of Ansible on RHEL

Once the Ansible Core rpm installation completes in the system, it has the configuration file and binary files that are commonly used, as shown in Table 3-2.

*Table 3-2 List of files that are associated with the ansible-core rpm*

Important and executable files	Description
<code>/etc/ansible/ansible.cfg</code>	The default configuration file that comes with Red Hat Package Manager packages. It has all the required settings, the location of the module search path, module, executable files, and inventory file.
<code>/etc/ansible/hosts</code>	A sample inventory for the managed node or target host where automation tasks run.
<code>/usr/bin/ansible-config</code>	This command shows the effective Ansible configuration details and the file location. Which configuration file is used depends on the file location. Here is the order of importance of the files: <ul style="list-style-type: none"><li>▶ <code>/etc/ansible/ansible.cfg</code>: The default configuration file, which is used if it is present.</li><li>▶ <code>~/.ansible.cfg</code>: The user configuration file, which overrides the default configuration file.</li><li>▶ <code>./ansible.cfg</code>: A local configuration file that is in the current working directory. The file is assumed to be project-specific and overrides the other files.</li><li>▶ <code>ANSIBLE_CONFIG</code>: Specifies the override location for the Ansible config file.</li></ul> For example, the following command creates a sample configuration for you: <code># ansible-config init --disabled -t all &gt; ansible.cfg</code>
<code>/usr/bin/ansible</code>	This command defines and runs a single task 'playbook' against a set of hosts. For example, run a shell module to run a command: <code># ansible all -i hosts -m shell -a "hostname"</code>
<code>/usr/bin/ansible-console</code>	This command dynamically runs Ansible modules or arbitrary commands to the hosts. Here is an example: <code># ansible-console -i hosts --limit all -u root</code>

Important and executable files	Description
/usr/bin/ansible-doc	This command shows information about specific modules that are installed in Ansible libraries. For example, to check the copy module documentation, run the following command: # ansible-doc copy
/usr/bin/ansible-playbook	This command runs Ansible playbooks to run the defined tasks on the targeted hosts. For example, to run a sample playbook, run the following command: # ansible-playbook myplaybook.yml
/usr/bin/ansible-vault	This command can be used as an encryption/decryption utility for Ansible that can encrypt any structured data file that is used by Ansible.

To continue with the verification and configuration, complete the following steps:

1. Generate the configuration file by running the following command:  
# ansible-config init --disabled -t all > ansible.cfg
2. Display the effective configuration and its configuration file location, in terms of the current working location or directory, as shown in Example 3-4.

*Example 3-4 Displaying the effective configuration*

---

```
# ansible-config --version
ansible-config [core 2.14.2]
  config file = /root/ansible.cfg
  configured module search path = ['/root/.ansible/plugin/modules',
'/usr/share/ansible/plugin/modules']
  ansible python module location = /usr/lib/python3.11/site-packages/ansible
  ansible collection location =
/root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible-config
  python version = 3.11.2 (main, Jun 6 2023, 07:39:01) [GCC 8.5.0 20210514
(Red Hat 8.5.0-18)] (/usr/bin/python3.11)
  Jinja version = 3.1.2
  libyaml = True
```

---

3. Verify the inventory file name and location, as shown in Example 3-5.

*Example 3-5 Verifying the inventory file location*

---

```
# grep -vE "^#|^;" /root/ansible.cfg|grep -v ^$
[defaults]
inventory=./hosts
[privilege_escalation]
[persistent_connection]
[connection]
[colors]
[selinux]
[diff]
[galaxy]
[inventory]
[netconf_connection]
[paramiko_connection]
[jinja2]
[tags]
```

```
[runas_become_plugin]
[su_become_plugin]
[sudo_become_plugin]
[callback_tree]
[ssh_connection]
[winrm]
[inventory_plugins]
[inventory_plugin_script]
[inventory_plugin_yaml]
[url_lookup]
[powershell]
[vars_host_group_vars]
```

---

4. Verify the inventory file configuration, as shown in Example 3-6.

*Example 3-6 Verifying the inventory configuration*

---

```
# cat /etc/ansible/hosts
192.168.121.203
```

---

5. Run an ad hoc command to verify the function, as shown in Example 3-7.

*Example 3-7 Testing the Ansible function with an ad hoc command*

---

```
# ansible all -i hosts -m shell -a "hostname" -u root -k
SSH password:
192.168.121.203 | CHANGED | rc=0 >>
localhost.localdomain
```

---

## Additional preparation and configuration for Ansible on RHEL

A best practice is to create a separate user to manage automation activities, generate ssh keys for managed nodes access, create a virtual environment for specific Python versions, and set environment variables for a better working environment. To do so, complete the following steps:

1. Create a user who is called `ansible` by using the following command:

```
# useradd -m -c "Ansible Controller User" ansible
```

2. Install any additional Python libraries or modules depending on your requirements by using the following command:

```
# dnf install python3-pyOpenSSL python3-winrm python3-netaddr python3-psutil
python3-setuptools
```

**Note:** If any specific Python libraries or modules are not available or not shipped with RHEL OS in rpm format, they can be installed by using `pip` (the Python packages manager). You can create a virtual environment like a virtual machine (VM) or Linux `chroot` that has an isolated structure of lightweight directories that are separated from the OS Python directories to use different versions of Python modules, files, or configurations.

3. Create a `~/ .vimrc` file to customize the vim editor configuration to use the 2 space indentation for yam1 file editing, as shown in Example 3-8.

*Example 3-8 Modifying the vim editor configuration*

```
# cat << 'EOF' >> ~/ .vimrc
autocmd FileType yam1 setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
EOF
```

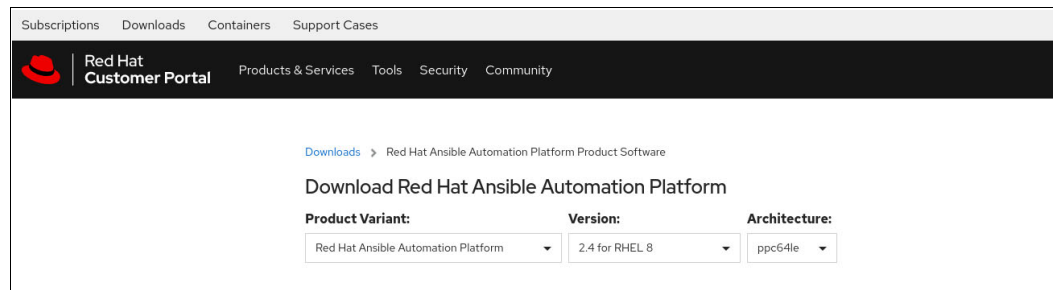
4. Generate and copy the SSH key from the Ansible Automation Controller node to managed nodes by using the following commands:

```
# ssh-keygen
# ssh-copy-id root@192.168.121.203
```

## Ansible Automation Platform installation on RHEL

The Ansible Automation Platform can be installed and configured in IBM Power servers by completing the following steps:

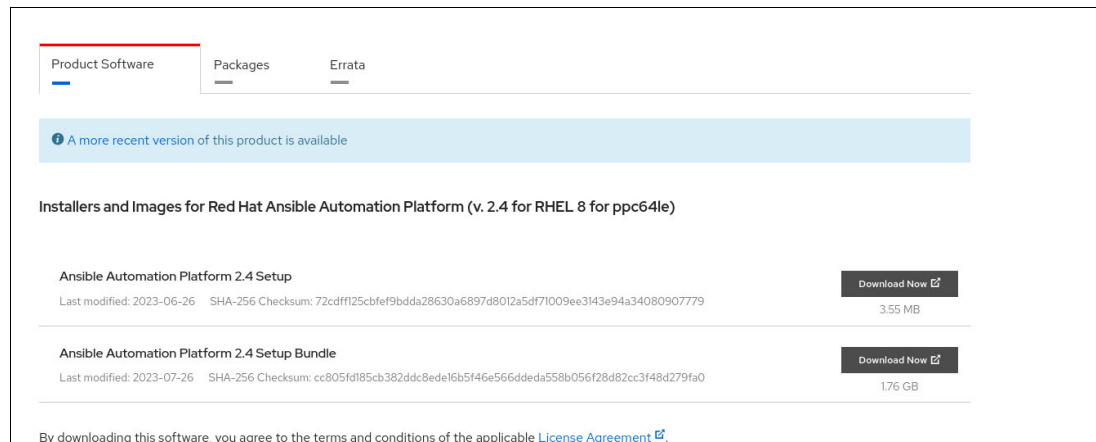
1. Download the Ansible Automation Platform installer from the [Red Hat product download site](#). From the Red Hat product download site, select the product that is named “Red Hat Ansible Automation Platform”, as shown in Figure 3-13.



*Figure 3-13 Selecting the version and architecture for the Ansible Automation Platform package download*

The download list is available once you select the Ansible Automation Platform version and the architecture from the product software download page.

The software download page is shown in Figure 3-14.



*Figure 3-14 List of Ansible Automation Platform package bundles that can be downloaded*

Download the bundle package. Here is an example file name:

ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le.tar.gz

2. Copy that tar file to the system and extract the files, as shown in Example 3-9.

*Example 3-9 Copying and extracting the file*

---

```
# tar xvzf ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le.tar.gz
# ls -l
::::::::::::Some Output Removed::::::::::::
drwxrwxrwx. 5 root root      4096 Jul 23 10:58
ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le
-rw-r--r--. 1 root root 2068376768 Jul 23 00:06
ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le.tar.gz
::::::::::::Some Output Removed::::::::::::
```

---

3. Go to the extracted directory and configure the inventory file by using a vim editor for the all-in-one installation scenario. This process is shown in Example 3-10.

*Example 3-10 Configuring the inventory file*

---

```
# cd ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le/
# ls -l
::::::::::::Some Output Removed::::::::::::

-rw-rw-rw-. 1 root root    530 Jun 26 19:55 README.md
drwxrwxrwx. 5 root root   4096 Jun 26 19:43 bundle
drwxrwxrwx. 3 root root   4096 Jun 26 19:38 collections
drwxrwxrwx. 2 root root   4096 Jun 26 19:38 group_vars
-rw-rw-rw-. 1 root root   8653 Jul 23 10:30 inventory
-rwxrwxrwx. 1 root root  14780 Jun 26 19:38 setup.sh

# vim inventory
# grep -v ^# inventory |grep -v ^$
[automationcontroller]
bs-rbk-lnx-1.power-iaas.cloud.ibm.com node_type=hybrid
[automationcontroller:vars]
peers=execution_nodes
[execution_nodes]
[automationhub]
[automationedacontroller]
[database]
[sso]
[all:vars]
admin_password='Redhat123'
pg_host=''
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='Redhat123'
pg_sslmode='prefer' # set to 'verify-full' for client-side enforced SSL
registry_url='registry.redhat.io'
registry_username=''
registry_password=''
receptor_listener_port=27199
automationedacontroller_admin_password=''
automationedacontroller_pg_host=''
```

```

automationedacontroller_pg_port=5432
automationedacontroller_pg_database='automationedacontroller'
automationedacontroller_pg_username='automationedacontroller'
automationedacontroller_pg_password=''
sso_keystore_password=''
sso_console_admin_password=''

```

**Note:** The legacy execution environment (ee\_29\_enabled=true) is not supported for Power servers. If ee\_29\_enabled = true is enabled, then you receive the errors that are shown in Figure 3-15.

```

TASK [ansible.automation_platform_installer.preflight : Check the architecture for ee-29 container image] ***
fatal: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]: FAILED! => {
  "assertion": "ansible_architecture == 'x86_64'",
  "changed": false,
  "evaluated_to": false,
  "msg": "The ee-29 container image is only available on x86_64 architecture"
}
NO MORE HOSTS LEFT *****

```

Figure 3-15 Error that is caused by ee\_29\_enabled = true

4. Run the Ansible Automation Platform setup script to start the installation, as shown in Figure 3-16.

```

oryPreserve": "no", "RuntimeMaxUsec": "infinity", "SameProcessGroup": "no", "SecureBits": "0", "SendSIGHUP": "no", "SendSIGKILL": "yes", "Slice": "system.slice",
"StandardError": "inherit", "StandardInput": "null", "StandardInputData": "", "StandardOutput": "journal", "StartLimitAction": "none", "StartLimitBurst":
"5", "StartLimitIntervalUsec": "10s", "StartupBlockIOWeight": "[not set]", "StartupCPUShares": "[not set]", "StartupCPUWeight": "[not set]", "StartupIOWeight":
"[not set]", "StateChangeTimestamp": "Wed 2023-08-23 10:16:14 EDT", "StateChangeTimestampMonotonic": "1036299288", "StateDirectoryMode": "0755", "StatusErr
no": "0", "StopWhenUnneeded": "no", "SubState": "dead", "SuccessAction": "none", "SyslogFacility": "3", "SyslogLevel": "6", "SyslogLevelPrefix": "yes", "Syslo
gPriority": "30", "SystemCallErrorNumber": "0", "TTYReset": "no", "TTYVHangup": "no", "TTYVDiallocate": "no", "TasksAccounting": "yes", "TasksCurrent": "[no
t set]", "TasksMax": "97196", "TimeoutStartUsec": "1min 30s", "TimeoutStopUsec": "1min 30s", "TimerSlackNSec": "50000", "Transient": "no", "Type": "forking",
"UID": "[not set]", "UMask": "0022", "UnitFilePreset": "disabled", "UnitFileState": "enabled", "UtmpMode": "init", "WantedBy": "automation-controller.service
multi-user.target", "WatchdogTimestampMonotonic": "0", "WatchdogUsec": "0"}}

PLAY [Post-install insights setup] *****
TASK [include_role : ansible.automation_platform_installer.misc] *****
TASK [ansible.automation_platform_installer.misc : Ensure insights-client is installed] ***
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => {"changed": false, "msg": "Nothing to do", "rc": 0, "results": []}

TASK [ansible.automation_platform_installer.misc : Register with Insights] *****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => {"changed": true, "cmd": ["insights-client", "--register"], "delta": "0:01:38.333430", "end": "2023-08-23
10:18:08.268092", "msg": "", "rc": 0, "start": "2023-08-23 10:16:29.934662", "stderr": "", "stderr_lines": [], "stdout": "Successfully registered host bs-rbk-
lnx-1.power-iaas.cloud.ibm.com\nAutomatic scheduling for Insights has been enabled.\nStarting to collect Insights data for bs-rbk-lnx-1.power-iaas.cloud.ibm.c
om\nUploading Insights data.\nSuccessfully uploaded report from bs-rbk-lnx-1.power-iaas.cloud.ibm.com to account 5910538.\nView the Red Hat Insights console a
t https://console.redhat.com/insights/", "stdout_lines": ["Successfully registered host bs-rbk-lnx-1.power-iaas.cloud.ibm.com", "Automatic scheduling for Ins
ights has been enabled.", "Starting to collect Insights data for bs-rbk-lnx-1.power-iaas.cloud.ibm.com", "Uploading Insights data.", "Successfully uploaded rep
ort from bs-rbk-lnx-1.power-iaas.cloud.ibm.com to account 5910538.", "View the Red Hat Insights console at https://console.redhat.com/insights/"]}

PLAY [Post-install cleanup] *****
TASK [Remove stale packages] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => {"changed": false, "msg": "Nothing to do", "rc": 0, "results": []}

PLAY RECAP *****
bs-rbk-lnx-1.power-iaas.cloud.ibm.com : ok=380 changed=177 unreachable=0 failed=0 skipped=207 rescued=0 ignored=6
localhost : ok=0 changed=0 unreachable=0 failed=0 skipped=1 rescued=0 ignored=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2023-08-23-09:56:26.log.
[root@bs-rbk-lnx-1 ansible-automation-platform-setup-bundle-2.4-1.2-ppc64le]# █

```

Figure 3-16 Screen capture from the installation script

**Note:** The default minimum RAM size is 8 GiB. This value can be modified for a non-production or a testing environment by changing the default configuration file at the following location:

```
collections/ansible_collections/ansible/automation_platform_installer/roles/preflight/defaults/main.yml
```

To adjust the minimum RAM size, modify the required\_ram entry before continuing the installation. For example:

```
required_ram: 4000
```



5. Once installation successfully completes, log in to the Ansible Automation Platform UI, as shown in Figure 3-17.

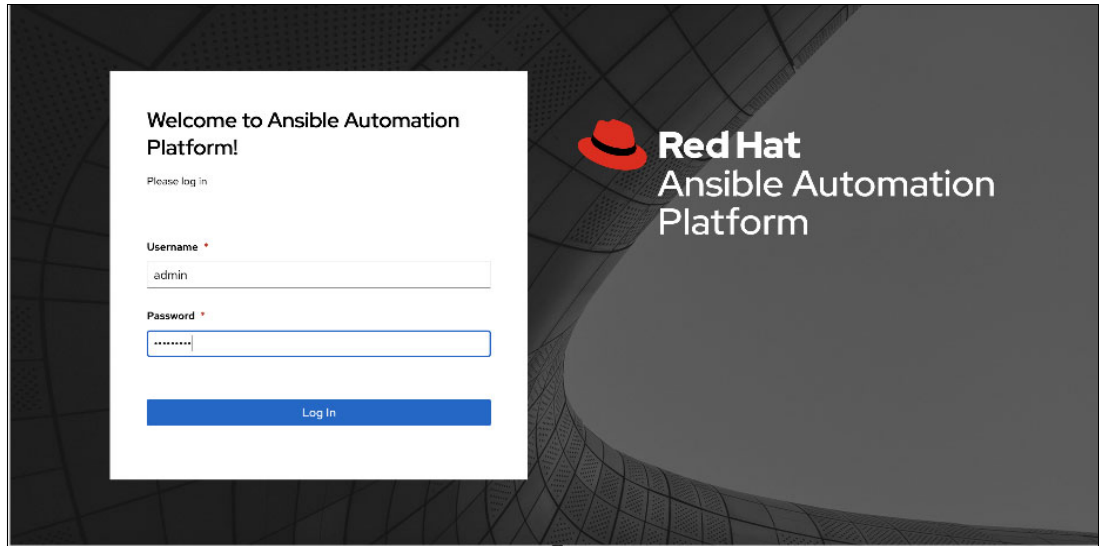


Figure 3-17 Ansible Automation Platform login page

6. When you log in the first time, you must configure the subscription manager and activate your subscription. In a disconnected or restricted environment (that is, no internet access from the system), you must first create a manifest file, allocate the Red Hat software subscriptions with Ansible Automation Platform to the manifest, and then export the manifest to enable you to download the manifest file that you created.

Uploading the manifest is shown in Figure 3-18.

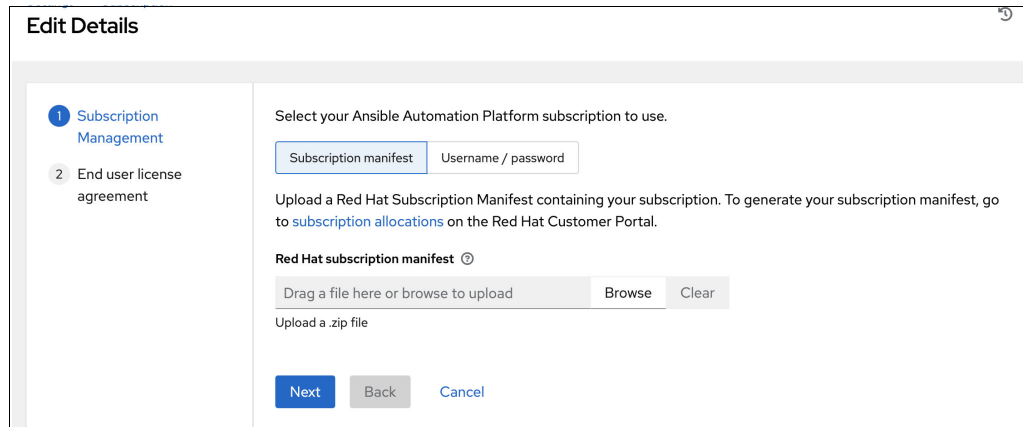


Figure 3-18 Ansible Automation Platform subscription activated by using a manifest file

For more information about creating and using a Red Hat Satellite manifest, see [How to create and use a Red Hat Satellite manifest](#).

If the system is directly connected to the internet, you can use a Red Hat software subscription username and password for the activation, as shown in Figure 3-19.

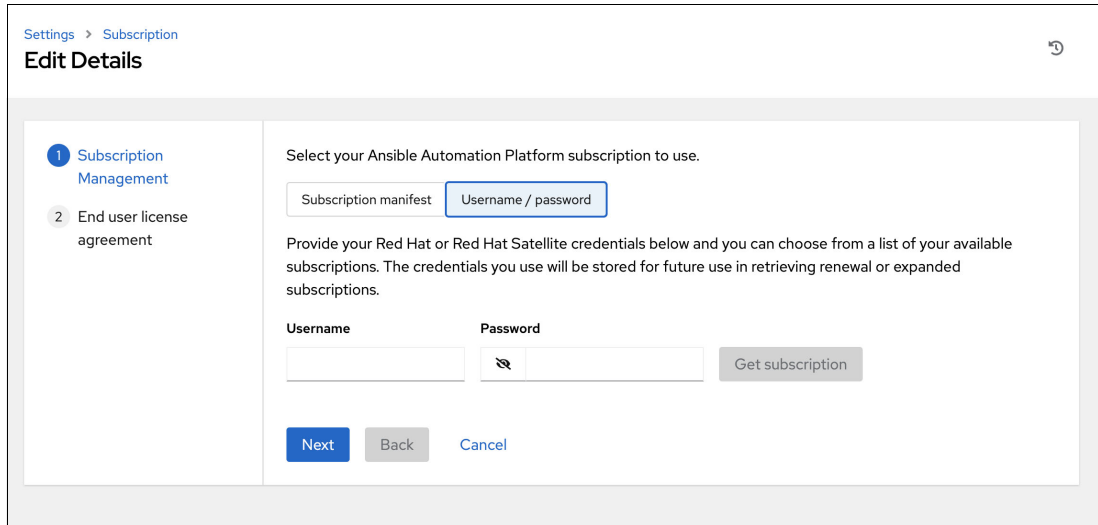


Figure 3-19 Ansible Automation Platform subscription activation by using a username and password

7. Once you log in, select the appropriate subscription from the list. An example is shown in Figure 3-20. Click the **Next** on the User and Automation Analytics window, and then click the **Submit** on the End User License Agreements window.

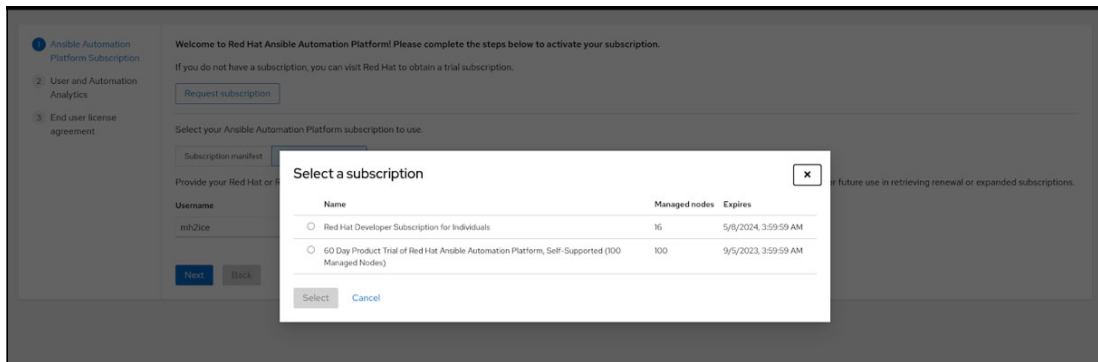


Figure 3-20 Selecting a subscription for installation

Ansible Automation Platform is ready for further integration and configuration for you to start automating your environment, as shown in Figure 3-21 on page 131.

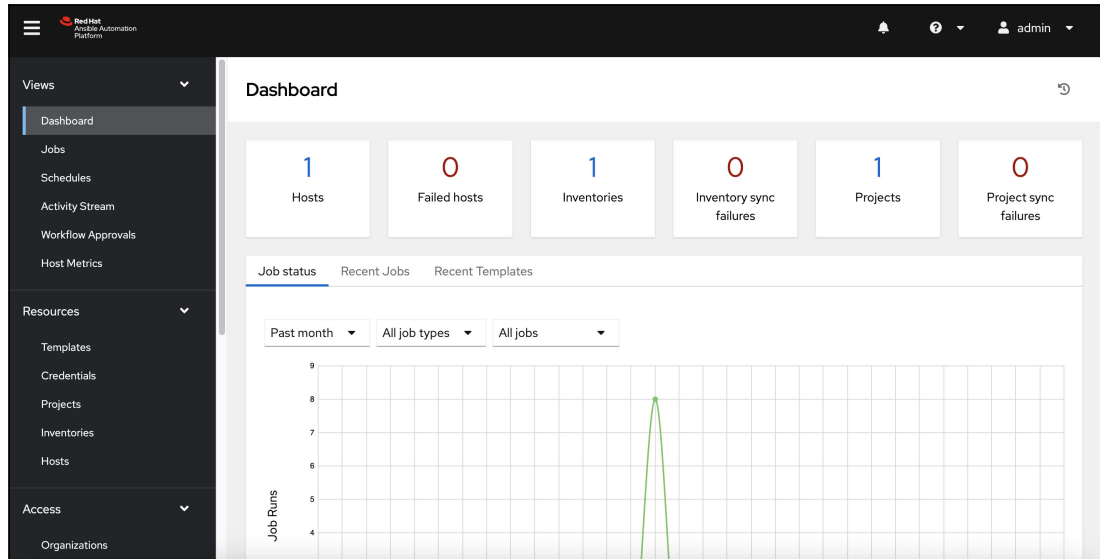


Figure 3-21 Ansible Automation Platform dashboard window

### Further configuration steps

Now that Ansible Automation Platform is installed, it can be used to configure the required integrations and required resources for your automation projects by using a web console.

Here are some of the resources that can be created in the Ansible Automation Platform:

- ▶ Templates (See [Job Templates](#) and [Workflow Job Templates](#).)
- ▶ Credentials
- ▶ Projects
- ▶ Inventories
- ▶ Hosts
- ▶ Organizations
- ▶ Users
- ▶ Teams

Also, you can configure and integrate third-party services that you require. Here are some example services:

- ▶ Enhanced and simplified role-based access control (RBAC) and auditing: Configure RBAC. You can use an automation controller to grant permissions to perform a specific task (such as to view, create, or modify a file) to different teams or explicit users through RBAC.
- ▶ Backup and restore: The ability to back up and restore your system is integrated into the Ansible Automation Platform setup playbook. You can configure cron jobs and use a `setup.sh` script for backup and restore.
- ▶ Integrated notifications: Configure stackable notifications for job templates, projects, or entire organizations, and configure different notifications for job start, job success, job failure, and job approval (for workflow nodes). Notifications can be integrated with email, Grafana, Slack, or other tools.
- ▶ Authentication enhancements: The automation controller supports LDAP, SAML, and token-based authentication. Configure a feasible authentication method.

- ▶ **Workflow enhancements:** To better model your complex provisioning, deployment, and orchestration workflows, the automation controller expanded workflows in many ways:
  - Inventory overrides for workflows
  - Convergence nodes for workflows
  - Workflow nesting
  - Workflow pause and approval
- ▶ **Secret management system:** With a secret management system, external credentials are stored and supplied for use in the automation controller.
- ▶ **Manage playbooks by using source control:** Manage playbooks and playbook directories by either placing them manually under the project or placing the playbooks into a supported SCM system, including Git, Subversion, and Mercurial. Configuration and integration with SCM.

For more information post-configuration, see the [Automation Controller User Guide v4.4](#).

## **Ansible Automation Controller installation on RHEL**

The heart of automation is the Ansible Automation Controller, which is a system that runs the Ansible commands and playbooks in a deterministic way to automate your environment.

### ***System preparation on RHEL 8***

Create a user (for example, `ansible`) that owns the environment and installs the necessary packages on the OS. Also, avoid installing `pip` packages outside a Python virtualenv so that the OS managed Python modules that are installed for other uses are not interfered with based on the Python requirements of your Ansible installation.

Also, modify your VIM configuration to replace “Tab” with an indent that uses “2 white spaces”.

Run the commands that are shown in Example 3-11 as root.

#### *Example 3-11 Installing Ansible*

---

```
useradd -m -c "Ansible Controller venv User" ansible
dnf install ansible-core python3-virtualenv vim
```

```
cat << 'EOF' > /etc/pip.conf
[install]
require-virtualenv = true
```

```
[uninstall]
require-virtualenv = true
EOF
```

---

### ***Create the virtual environment***

To create the virtual environment, complete the following steps:

1. Create the virtual environment by running the following command:
 

```
virtualenv --python='/usr/bin/python3.9' ~/venv
```
2. Upgrade `pip`, and other necessary Python libraries, and install the requirements for the most used Ansible collections, as shown in Example 3-12 on page 133.

### Example 3-12 Installing the Python libraries

---

```
python3.11 -m venv ~/venv
source ~/venv/bin/activate
export PYTHONPATH=$( ls -ld ~/venv/lib/python*/site-packages )
pip install -U pip setuptools psutil
pip install jmespath netaddr pywinrm pypsrp pyopenssl
```

```
ansible-galaxy collection install community.general ansible.windows ansible.posix
ansible.utils
```

---

3. Create a default `ansible.cfg` file for this environment, configure the default `hosts.ini` file, and populate it with at least the Ansible Controller itself (localhost), as shown in Example 3-13.

### Example 3-13 Creating the `ansible.cfg` file

---

```
ansible-config init --disabled -t all > ~/ansible.cfg
perl -pi -e "s|^\;?(inventory=).*|\1~/hosts.ini|g" ~/ansible.cfg
```

```
cat << 'EOF' >> ~/hosts.ini
localhost ansible_connection=local
EOF
```

---

4. To make it convenient to get into the virtualenv while logging in as user `ansible`, add the source and export lines to the `.bashrc` (or `.profile`) of the user.
5. Adjust `vimrc` to make VIM recognize the `yaml/yml` indentation, as shown in Example 3-14.

### Example 3-14 Adjusting VIM for `yaml` notation

---

```
cat << 'EOF' >> ~/.vimrc
autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
EOF
```

---

6. Adjust `.bashrc` to reflect the environment that is shown in Example 3-15.

### Example 3-15 Adjusting `.bashrc` for Python

---

```
~/ .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User-specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

export HISTSIZE=100000 # big big history
export HISTFILESIZE=100000 # big big history
```

```
# Uncomment the following line if you don't like systemctl's auto-paging feature:
```

```
# export SYSTEMD_PAGER=

# User specific aliases and functions
alias view="vim -R"

source ~/ansible-venv/bin/activate
export PYTHONPATH=$( ls -ld ~/ansible-venv/lib/python*/site-packages )
```

---

Your virtual environment is now ready to use.

### 3.3.2 AIX as an Ansible Controller

There are multiple implementations of Ansible that can be used, depending on your requirements and your environment. Ansible Core is the supported implementation for AIX because Ansible Automation Platform is not supported on AIX. To get the full benefit of Ansible Automation Platform running on your IBM Power server, choose an LPAR running RHEL as your controller.

If you installed open-source tools on AIX, installing Ansible will be familiar and look similar to an installation on a Linux on Power LPAR. However, if you have limited experience with open-source deployments on AIX, you must learn the open-source installation methodology.

Ansible on AIX is delivered as a part of [IBM AIX Toolbox for Open Source Software](#). All software that is delivered through IBM AIX Toolbox for open-source applications is packaged by using the Red Hat Package Manager format, which is the same format that is used in Linux. Because the format is not the AIX native BFF package format, the installation procedure is different.

It is a best practice to use `dnf` to install any open-source tools in your AIX environment. The `dnf` command is in a package manager for Red Hat Package Manager packages. It is an updated version of the `yum` command, which you might see in a Linux environment. Although you can install Red Hat Package Manager files without a package manager, using `dnf` has two significant advantages: It can automatically resolve dependencies and then install them from package repositories. Without the `dnf` package manager, you are forced to manually determine any package dependencies and then install them. You can find a useful and detailed guide about installing `dnf` at [DNF is now available on AIX Toolbox](#).

In our testing scenarios, we ran the installations on both AIX 7.2 and AIX 7.3. The process applies to both versions.

#### Installing on a system with internet connectivity

If your AIX server has a direct internet connection, running the `dnf_aixtoolbox.sh` script should install `dnf` on your machine. To download the script and run it, use the `/dnf_install.sh -y` command. It runs for a while and sets up `dnf` if everything is successful.

Example 3-16 shows the steps to install the package.

*Example 3-16 Installing DNF on AIX*

---

```
# /usr/opt/perl5/bin/lwp-download
https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/dnf_aixtoolbo
x.sh
# chmod +x dnf_aixtoolbox.sh
# ./dnf_aixtoolbox.sh -y
```

---

The script downloads the newest rpm.rte package and a bundle of Red Hat Package Manager packages to install. There are many packages in the bundle, but the most important in this case are Python 3.9 and DNF itself.

## Installing on a system without internet connectivity

If you do not have an internet connection, then some extra steps are required to install dnf. To successfully install the *dnf* package, complete the following steps:

1. Verify that you have your proxy set up correctly and export the variables, as shown in Example 3-17.

*Example 3-17 Setting up the proxy variables*

---

```
export http_proxy=http://user:password@IP:PORT/  
export https_proxy=http://user:password@IP:PORT/  
Your proxy setup should look something like  
export http_proxy=http://atilio:b01s111ud0@192.168.0.45:8080/
```

---

2. The command to run the script is `/dnf_install.sh -y`. The script requires FTP access to IBM. If you cannot support FTP, comment out lines 179 - 254 in the script and download the packages manually from one of the following repositories:

[https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf\\_bundle\\_aix\\_71\\_72.tar](https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf_bundle_aix_71_72.tar)  
[https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf\\_bundle\\_aix\\_73.tar](https://public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/eziinstall/ppc/dnf_bundle_aix_73.tar)

3. Check your OpenSSL version. It must be at least Version 1.1. In our example, we used `openssl-1.1.2.2200.tar.Z`, which we downloaded from the following website:

[https://www.ibm.com/resources/mrs/assets/DirectDownload?source=aixbp&lang=en\\_US](https://www.ibm.com/resources/mrs/assets/DirectDownload?source=aixbp&lang=en_US)

**Note:** You need an IBMid to download this file.

4. If the installation stalls, the rpm.rte installation might be failing. Open the `.tar` file, extract rpm.rte, and update it by using `smit` or `installp` from the CLI.

## Alternative installation steps for systems without internet connectivity

If you do not have internet access on your IBM AIX server and do not want to complete the steps in “Installing on a system without internet connectivity”, download the latest bundle manually to any server and then transfer it to your AIX server.

You can find the latest bundles [here](#). Complete the following steps:

1. There are two bundles: one for AIX 7.1 and 7.2, and another one for AIX 7.3. Choose the correct bundle for your version of IBM AIX.
2. After downloading the bundle, extract it to a temporary directory.
3. In the temporary directory, you find the script `dnf_install.sh`. Run `./dnf_install.sh -y`. It sets up the dnf module.

## Steps to take after the installation of dnf

After you install dnf, complete the following steps:

1. Update the packages by running `dnf -y update`.
2. In the default configuration, dnf tries to download package information from IBM. If you work in an air-gapped environment without direct access to the internet, complete the following steps:
  - a. Create local repositories by mirroring IBM repositories.
  - b. After creating local mirrors, reconfigure dnf by manually editing `/opt/freeware/etc/dnf/dnf.conf`.

## Ansible installation on AIX

With dnf installed and setup, proceed to the installation of Ansible and the `ansible-core` packages.

If you search for Ansible in the repositories, you find three references to it, as shown in Example 3-18.

### *Example 3-18 Searching for Ansible in the AIX repositories*

---

```
# dnf search ansible
===== Name & Summary Matched: ansible =====
ansible.noarch : Curated set of Ansible collections included in addition to
ansible-core
===== Name Exactly Matched: ansible =====
ansible.ppc : SSH-based configuration management, deployment, and task execution
system
===== Name Matched: ansible =====
ansible-core.noarch : A radically simple IT automation system
```

---

The package that you want to install is called `ansible.noarch`. The package `ansible-core.noarch` is the base package of Ansible, which provides Ansible Core 2.14.2 at the time of writing.

The package `ansible.noarch` provides extra collections that you usually need to work with Ansible. If you install the package `ansible.noarch`, it automatically installs the package `ansible-core.noarch`.

**Important:** Do not install `ansible.ppc` because it is an earlier version of Ansible.

Example 3-19 shows the command to install Ansible and the resulting output.

### *Example 3-19 Installing Ansible on IBM AIX*

---

```
#dnf -y install ansible.noarch
Dependencies resolved.
=====
Package                Architecture      Version
Repository             Size
=====
Installing:
ansible                noarch           7.2.0-1
AIX_Toolbox_noarch     47 M
Installing dependencies:
```



ansible-core	noarch	2.14.2-1
AIX_Toolbox_noarch	3.5 M	
python3.9-packaging	noarch	19.2-2
AIX_Toolbox_noarch	58 k	
python3.9-pyparsing	noarch	2.4.4-2
AIX_Toolbox_noarch	196 k	
python3.9-resolvelib	noarch	0.5.4-1
AIX_Toolbox_noarch	30 k	

Transaction Summary

=====

Install 5 Packages

**Postinstallation configuration suggestions**

Consider the following configuration suggestions for your Ansible environment:

1. All Red Hat Package Manager packages from AIX Toolbox applications are installed in /opt/freeware. Usually, this directory is not added to the PATH variable in /etc/environment or to the user's profile. To run the Ansible commands, specify the full path to the command, as shown in the following commands:

/opt/freeware/bin/ansible-playbook or /opt/freeware/bin/ansible-galaxy.

For your convenience, it is a best practice to add /opt/freeware/bin into the PATH variable in your profile, as shown in Example 3-20. After you change your profile, re-login to enable the changes.

*Example 3-20 Adding /opt/freeware/bin to the PATH variable*

```
# echo 'export PATH=$PATH:/opt/freeware/bin' >>~/.profile
```

2. The global configuration of Ansible can be found in /opt/freeware/etc/ansible. By default, this global configuration is used, but you can set up Ansible to use local project-specific configuration files.

The same situation applies to Ansible collections. You can install them globally into /usr/share/ansible/collections, locally for your user, or for one project.

**Note:** As a best practice, create configuration files and install collections on a project basis.

3. Make sure that the correct locale files are installed. Ansible requires the UTF - 8 locale. The command to validate your installed locale files is shown in Example 3-21.

*Example 3-21 Validating the locale files that are installed*

```
# ls -lpp -l | grep -i bos.loc
bos.loc.com.utf          7.2.0.0 COMMITTED Common Locale Support - UTF-8
bos.loc.utf.EN_US       7.2.0.0 COMMITTED Base System Locale UTF Code
# This is a sample for AIX 7.2 the important thing is the package version might change.
```

**Note:** Not having the correct locale file causes Ansible commands to fail with the following error:

```
# /opt/freeware/bin/ansible
ERROR: Ansible requires the locale encoding to be UTF-8; Detected ISO8859-1.
```

4. You need to set your user environment as shown in Example 3-22.

*Example 3-22 User environment setup for using Ansible*

---

```
# vi .profile
".profile" 8 lines, 309 characters
export PATH=$PATH:/opt/freeware/bin
export TERM=aixterm
LC_MESSAGES=%l.%c
export LC_MESSAGES
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%l.%c/%N:/usr/lib/nls/msg/%L/%N.
cat:/usr/lib/nls/msg/%l.%c/%N.cat:/usr/lib/nls/msg/%l.%c/%N:/usr/lib/nls/msg/%l.
%c/%N.cat
export NLSPATH
LANG=EN_US
export LANG
```

---

### **Running a validation command**

When you are done with all the steps in “Postinstallation configuration suggestions” on page 137, you can run an Ansible command to validate that the installation of Ansible completed successfully. You can run the command that is shown in Example 3-23 to validate the installation and display the version of Ansible that is installed.

*Example 3-23 The # ansible --version command*

---

```
# ansible --version
ansible [core 2.14.2]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/usr/share/ansible/plugin-modules',
  '/usr/share/ansible/plugin-modules']
  ansible python module location =
  /opt/freeware/lib/python3.9/site-packages/ansible
  ansible collection location =
  /.ansible/collections:/usr/share/ansible/collections
  executable location = /opt/freeware/bin/ansible
  python version = 3.9.16 (main, Jun 28 2023, 12:45:03) [GCC 8.3.0]
  (/opt/freeware/bin/python3.9)
  Jinja version = 3.0.3
  libyaml = True
```

---

As you can see from the example, Ansible Core 2.14.2 was installed and the Python version is Python3 3.9.16.

### **Files that are installed during the Ansible installation on AIX**

When the Ansible Core rpm installation completes in the system, the system has the configuration file and binary files. The commonly used files are shown in Table 3-3 on page 139.

Table 3-3 List of files that associated with *ansible-core rpm*

Important files and executable files	Description
/etc/ansible/ansible.cfg	The default configuration file that comes with Red Hat Package Manager packages. It has all the required settings, the location of the module search path, and the module, executable files, and inventory files. The configuration file has precedence based on its location (linked to /opt/freeware/etc/ansible/ansible.cfg).
/etc/ansible/hosts	A sample inventory for the managed node or target host where automation tasks run (linked to /opt/freeware/etc/ansible/hosts).
/opt/freeware/bin/ansible-config	Use this command to view the effective Ansible configuration details and the file location. The configuration file has precedence based on its location: <ul style="list-style-type: none"> <li>▶ /etc/ansible/ansible.cfg: The default configuration file. It is used if it is present.</li> <li>▶ ~/.ansible.cfg: The user configuration file. It overrides the default configuration file.</li> <li>▶ ./ansible.cfg: A local configuration file that is in the current working directory. It is assumed to be project-specific and overrides all other files.</li> <li>▶ ANSIBLE_CONFIG: This file specifies the override location for the ansible-config file.</li> </ul> For example, the following command can create a sample configuration for you: <pre># ansible-config init --disabled -t all &gt; ansible.cfg</pre>
/opt/freeware/bin/ansible	Use this command to define and run a single task playbook against a set of hosts. For example, run the shell module to run a command: <pre># ansible all -i hosts -m shell -a "hostname"</pre>
/opt/freeware/bin/ansible-console	Use this command to dynamically run Ansible modules or arbitrary commands to the hosts. For example: <pre># ansible-console -i hosts --limit all -u root</pre>
/opt/freeware/bin/ansible-doc	Use this command to display information about specific modules that are installed in Ansible libraries. For example, to check the copy module documentation, run the following command: <pre># ansible-doc copy</pre>
/opt/freeware/bin/ansible-playbook	Use this command to run Ansible playbooks, which run the defined tasks on the targeted hosts. For example, to run a sample playbook, run the following command: <pre># ansible-playbook myplaybook.yml</pre>
/opt/freeware/bin/ansible-vault	Use this command as an encryption/decryption utility for Ansible that can encrypt any structured data file that is used by Ansible.

## Next steps

Now that Ansible is installed, set up a configuration for your environment by completing the following steps:

1. Generate the configuration file by running the following command:

```
# ansible-config init --disabled -t all > ansible.cfg
```

2. Display the effective configuration and the configuration file location in terms of the current working location or directory, as shown in Example 3-24.

*Example 3-24 Displaying the effective configuration*

---

```
# ansible --version
ansible [core 2.14.2]
  config file = /ansible.cfg
  configured module search path = [!/.ansible/plugin/modules',
!'/usr/share/ansible/plugin/modules']
  ansible python module location =
/opt/freeware/lib/python3.9/site-packages/ansible
  ansible collection location =
/.ansible/collections:/usr/share/ansible/collections
  executable location = /opt/freeware/bin/ansible
  python version = 3.9.16 (main, Jun 28 2023, 12:45:03) [GCC 8.3.0]
(/opt/freeware/bin/python3.9)
  Jinja version = 3.0.3
  libyaml = True
```

---

3. Verify the inventory file name and location, as shown in Example 3-25.

*Example 3-25 Verifying the inventory file location*

---

```
# grep -vE "^\|^;" /etc/ansible/ansible.cfg|grep -v ^$
[defaults]
[privilege_escalation]
[persistent_connection]
[connection]
[colors]
[selinux]
[diff]
[galaxy]
[inventory]
[netconf_connection]
[paramiko_connection]
[jinja2]
[tags]
[runas_become_plugin]
[su_become_plugin]
[sudo_become_plugin]
[callback_tree]
[ssh_connection]
[winrm]
[inventory_plugins]
[inventory_plugin_script]
[inventory_plugin_yaml]
[url_lookup]
[powershell]
[vars_host_group_vars]
```

---

4. Verify the inventory file configuration, as shown in Example 3-26.

*Example 3-26 Verifying the inventory configuration*

---

```
atilio-ansiblerh73: /> cat /etc/ansible/hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or IP addresses
# - A hostname/ip can be a member of multiple groups
[linux]
ansible-AAP-redbook
hugo-rhel8-ansible
vasco-rhel8-ansible
revelez-rhel9-ansible
[aix]
vitorio-ansibleaix72
atilio-ansibleaix73
```

---

5. Run an ad hoc command to verify the function, as shown in Example 3-27.

*Example 3-27 Testing the Ansible function with an ad hoc command (sshpass must be installed with dnf)*

---

```
atilio-ansiblerh73: /> ansible all -i hosts -m shell -a "hostname" -u root -k
SSH password:
hugo-rhel8-ansible | CHANGED | rc=0 >>
hugo-rhel8-ansible
```

---

## Additional preparation and configuration of Ansible on AIX

To manage automation activities, create a separate user to do the following activities:

- ▶ Generate SSH keys for managed nodes access.
- ▶ Create a virtual environment for specific Python versions.
- ▶ Set environment variables for a better working environment.

To set up the user (in our example, `ansible`), complete the following steps:

1. Create the user by running the following command:

```
# vitorio-ansibleaix72: /> mkuser -a "gecos=Ansible Controller User" ansible
```

2. Install any additional Python libraries or modules depending on your requirements by running the following command:

```
# dnf install python3-pip
```

**Note:** If any specific Python libraries or modules are not available or not included with the AIX OS in rpm format, they can be installed by using `pip` (the Python packages manager). You can create a virtual environment, like a VM or Linux `chroot`, that has an isolated structure of lightweight directories that is separated from the OS Python directories so that you can use different versions of Python modules, files, or configurations.

3. Create a `~/vimrc` file to customize the vim editor configuration to use the 2 space indentation for YAML file editing, as shown in Example 3-28.

*Example 3-28 Modifying the vim editor configuration*

---

```
# cat << 'EOF' >> ~/.vimrc
  autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab
  autocmd FileType yml setlocal ts=2 sts=2 sw=2 expandtab
  EOF
```

---

4. Generate and copy the ssh key from the Ansible Automation Controller node to the managed nodes by using the following commands:

```
# ssh-keygen
# ssh-copy-id root@192.168.xxx.xxx
```

Your AIX based Ansible Controller is ready for use for automation tasks.

## Installing the IBM Power AIX Collection

You learned about the IBM Power AIX collection in “IBM Power AIX collection” on page 41. The AIX collection contains many modules and roles to help you manage your IBM AIX LPARs. The collection can be installed on your Ansible Controller node by using the `ansible-galaxy` command, as shown in Example 3-29.

*Example 3-29 Installing ibm.power\_aix collection*

---

```
$ ansible-galaxy collection install ibm.power_aix
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ibm-power_aix-1.6.4.tar.gz to
/home/ansible/.ansible/tmp/ansible-local-13042144f894hz11/tmpvmhd3sw5/ibm-power_ai
x-1.6.4-x64201pn
Installing 'ibm.power_aix:1.6.4' to
'/home/ansible/.ansible/collections/ansible_collections/ibm/power_aix'
ibm.power_aix:1.6.4 was installed successfully
```

---

If you do not have direct access, you can download the collection on another server and then copy it to your Ansible Controller node, as shown in Example 3-30.

*Example 3-30 Installing ibm.power\_aix collection from a local file*

---

```
$ ls
ibm-power_aix-1.6.4.tar.gz
$ ansible-galaxy collection install ibm-power_aix-1.6.4.tar.gz
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Installing 'ibm.power_aix:1.6.4' to
'/home/ansible/.ansible/collections/ansible_collections/ibm/power_aix'
ibm.power_aix:1.6.4 was installed successfully
```

---

The collection documentation has a demonstration inventory file that you can view. However, for our test environment, our inventory file looks like what is shown in Example 3-31 on page 143.

*Example 3-31 Our inventory file for our test environment*

---

```
all: # keys must be unique, that is, only one 'hosts' per group
  hosts:
  vars:
  children: # key order does not matter, indentation does
    aix:
      children:
        nimserver:
          hosts:
            narancio-nim-master:
              ansible_host: narancio-nim-master
          vars:
            vm_targets: vitorio-ansibleaix72
        nimclient:
          hosts:
            vitorio-ansibleaix72:
              ansible_host: vitorio-ansibleaix72
            atilio-ansibleaix73:
              ansible_host: atilio-ansibleaix73
          vars:
            res_group: basic_res_grp
      hosts:
        cascarilla-ansibleaix73:
          ansible_host: cascarilla-ansibleaix73
  vios:
    hosts:
      gpc-s924-vios1:
        ansible_host: gpc-s924-vios1
    vars:
      res_group: vios_res_grp
```

---

***Using a NIM server as your Ansible Controller***

The Ansible Power AIX collection has many modules. As you look through the list of modules that are shown in Table 1-7 on page 41, you might notice that there are several that require a NIM server, which reinforces the best practice to run your Ansible Controller node for your AIX LPARs on your NIM server. When you run the NIM-based modules on your NIM server, then the Ansible Controller may run commands on the NIM clients, and already have the required contents to support upgrades or installations for your AIX Ansible Clients.

**3.3.3 IBM i as an Ansible Controller**

Ansible, a powerful automation tool, is enhanced on the IBM i platform through the integration of key components, each serving a crucial role in enabling automation processes.

**Portable Application Solutions Environment**

Portable Application Solutions Environment (PASE), when integrated within IBM i, offers a runtime environment that facilitates the running of chosen applications. This environment includes industry-standard and de facto standard shells, which establish a robust scripting platform. Functioning as an AIX release, PASE can be customized for communication with System Licensed Internal Code (SLIC) by using memory from SLIC, which is also used by the IBM Integrated Language Environment® (ILE). PASE and ILE work together on IBM i. However, Ansible requires a Python installation and operation from PASE, and playbooks require integration of Python commands.

## Red Hat Package Manager

Red Hat Package Manager hosts compiled binary files, with IBM crafting versions for the IBM i platform. Stored within Integrated File System (IFS) on IBM i, Red Hat Package Manager files adhere to the format <name>-<version>-<release>.<os>.<architecture>.rpm. A sample Red Hat Package Manager file name is `Ansible-2.9.9-1.ibm72.noarch.rpm`.

Red Hat Package Manager files for IBM i are accessible at [IBM i software](#).

## Yellowdog Updater, Modified

Yellowdog Updater, Modified (YUM), which is a no-charge, open-source utility, aids package management through CLI interactions. YUM enables the installation or updating of Red Hat Package Manager packaged software and handles dependencies within the Red Hat Package Manager package. IBM i uses YUM to install, update, upgrade, and remove packages.

Install YUM before initiating Ansible. This installation can be performed with or without an internet connection. To install YUM without an Internet connection, a one-time bootstrap process is used. For more information, see [YUM installation](#).

After YUM is installed, go to the directory where it is, run `cd /Qopensys/pkg/bin/`, and then run `yum -h`.

**Note:** Packages can be managed through SSH terminal commands or IBM Access Client Solution.

## SSHpass and ssh-keygen

The SSHpass package, non-interactive, acts as an Ansible Controller on IBM i. It simulates an interactive user entering the required SSH connection password. Use the following command to install SSHpass:

```
yum install sshpass
```

Also, ssh-keygen is a vital component of SSH. It generates a secure connection between the Ansible Controller and IBM i endpoint systems by employing a pair of keys: public and private. To generate these keys, run the following command:

```
ssh-keygen -t rsa
```

**Note:** By establishing these components and their integration, Ansible on IBM i gains a powerful foundation that is ready for versatile automation and management tasks.

## Ansible Controller installation on IBM i

Installing the Ansible Controller on the IBM i platform involves setting up the Red Hat Package Manager environment. Help ensure that Red Hat Package Manager is configured by following the steps that are outlined in the [IBM i Open Source documentation](#).



There are three methods for installing Ansible Controller:

- ▶ Installation without internet access: If your IBM i server lacks direct internet access, you can manually download the Red Hat Package Manager package from the [IBM repository](#). To facilitate this task, employ a proxy server on a local intranet. Configure YUM to use the proxy by editing the yum.conf file in /Q0penSys/etc/yum/yum.conf. Add the proxy settings as shown in Example 3-32.

*Example 3-32 Configuring a proxy for YUM*

---

```
[main]
proxy=http://proxy.mycompany.example.com:1234
proxy_username=user_name
proxy_password=passw0rd
```

---

You can create a local repository by using reposync and createrepo, which generate a complete copy of the remote repository. For more information about this process, see [Create a local repository mirror](#).

- ▶ Internet access from an IBM i server: If your IBM i server has internet access, help ensure that Python 3.6+ is installed. Log in to an SSH terminal and run `yum install ansible`, and then run `ansible -version`. The first command installs Ansible, and the second command verifies the installation by checking its version.
- ▶ Installation by using IBM i Access Client Solutions (ACS): Ansible can be conveniently installed through ACS. For step-by-step instructions, see [Installation using IBM i Access Client Solutions \(ACS\)](#). This approach offers a method to set up Ansible on your IBM i platform.

## Configuring Ansible for IBM i

The configuration of Ansible is a crucial step to help ensure its effective function. Here is a step-by-step guide about how to configure Ansible on your IBM i platform:

1. Locate the configuration file: The Ansible configuration file, typically named `ansible.cfg`, is usually in the `/etc/ansible` directory. However, you can also create this file in a different path or directory if needed.
2. Parameter flexibility: The `ansible.cfg` file encompasses a wide range of parameters that can be used. Activating all parameters is not mandatory, and you can configure them as required for your specific setup.
3. Creating the configuration file: To create the Ansible configuration file, run the following command:

```
vi /etc/ansible/ansible.cfg
```

In this file, you can specify various configurations to tailor Ansible's behavior to your needs. One of the commonly customized sections is `[defaults]`.

Example 3-33 shows how to configure the `ansible.cfg` file for IBM i.

*Example 3-33 Parameters for the Ansible configuration file for IBM i*

---

```
[defaults]
inventory =
~/ansible/collections/ansible_collections/ibm/power_ibmi/playbooks/hosts_ibmi.ini
library =
~/ansible/collections/ansible_collections/ibm/power_ibmi/plugin-ins/action
```

---

**Note:** Part of the Ansible configuration process involves setting up the `ansible.cfg` file. Although the default location for this file is typically `/etc/ansible` on various platforms, including Ansible Controller on IBM i, you must create one if it does not exist.

For a comprehensive list of parameters that can be included in an Ansible configuration file, see this [GitHub repository](#). This reference provides a deeper understanding of the various options that are available for configuring Ansible to suit your requirements.

## Postinstallation configuration steps on IBM i

After the installation of Ansible, further configuration is necessary to effectively use Ansible collections, modules, and automation capabilities on IBM i. Ansible collections represent the modern standard for distributing and maintaining automation components, encompassing modules, action plug-ins, roles, and sample playbooks.

**Note:** Installing collections with `ansible-galaxy` is supported only in Ansible 2.9+.

The following procedure guides you through the process of configuring Ansible Galaxy, which is a repository for Ansible on IBM i that contains content from the broader Ansible community:

1. Install the IBM i collection from Ansible Galaxy, which is the designated package manager for Ansible. Example 3-34 displays the command.

*Example 3-34 Command to install Ansible Galaxy collections*

---

```
# ansible-galaxy collection install ibm.power_ibmi
Process install dependency map
Starting collection install process
Installing 'ibm.power_ibmi:1.5.0' to
'/HOME/QSECOFR/.ansible/collections/ansible_collections/ibm/power_ibmi'
```

---

2. Check the installation path of the collections in the IFS by using the `cd` command:  
`cd /home/qsecofr/.ansible/collections/ansible_collections/ibm/power_ibmi`
3. Display the content of the `power_ibmi` directory by using the `ls -l` command, as shown in Example 3-35.

*Example 3-35 Displaying the content of the power\_ibmi directory*

---

```
# ls -l
total 220
-rw-r--r-- 1 qsecofr 0 90760 Jul 20 23:33 FILES.json
-rw-r--r-- 1 qsecofr 0 1241 Jul 20 23:33 MANIFEST.json
-rw-r--r-- 1 qsecofr 0 1284 Jul 20 23:33 README.md
-rw-r--r-- 1 qsecofr 0 186 Jul 20 23:33 binddep.txt
drwxr-sr-x 3 qsecofr 0 8192 Jul 20 23:33 changelogs
drwxr-sr-x 4 qsecofr 0 8192 Jul 20 23:33 docs
drwxr-sr-x 2 qsecofr 0 8192 Jul 20 23:33 meta
drwxr-sr-x 5 qsecofr 0 28672 Jul 21 12:20 playbooks
drwxr-sr-x 5 qsecofr 0 8192 Jul 20 23:33 plug-ins
drwxr-sr-x 24 qsecofr 0 28672 Jul 20 23:33 roles
drwxr-sr-x 9 qsecofr 0 12288 Jul 20 23:33 usecases
```

---

4. Go to the user's home directory by running the following command:

```
cd /home/qsecofr
```

5. Create a `.ssh` directory in the user's home directory by running the following command:

```
mkdir -p /home/qsecofr/.ssh
```

6. Verify the creation of the directory, as shown in Example 3-36.

*Example 3-36 Displaying the ssh directory*

---

```
# ls -la
total 56
drwxr-sr-x 5 qsecofr 0 8192 Sep 29 23:15 .
drwxrwsrwx 5 qsys   0 8192 Sep 20 2020 ..
drwx--S--- 3 qsecofr 0 8192 Sep 20 22:35 .ansible
drwxrwsrwx 3 qsecofr 0 8192 Oct 20 2019 .java
-rw-r--r-- 1 qsecofr 0 42 Oct 20 2019 .profile
drwxr-sr-x 2 qsecofr 0 8192 Sep 17 23:15 .ssh
-rw----- 1 qsecofr 0 16 Oct 20 2019 .vi_history
```

---

7. Generate an SSH key pair for the Ansible Controller on IBM i and its managed hosts. Enter the following command, pressing Enter three times without providing a passphrase or changing the default location of the key. The results are shown in Example 3-37.

*Example 3-37 Generating the RSA key pair*

---

```
# ssh-keygen -t rsa -C "10.10.10.10"
Generating public/private rsa key pair.
Enter the file in which to save the key (/HOME/QSECOFR/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter the same passphrase again:
Your identification has been saved in /HOME/QSECOFR/.ssh/id_rsa.
Your public key has been saved in /HOME/QSECOFR/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:V5VzRDMfVjLMFa7Wh8+V8q6e5IdQbFPys5HYc/tegIA 10.10.10.10
The key's randomart image is:
+---[RSA 3072]----+
|                 o+@B|
|                . oB+*|
|               E . o *+o|
|                o Bo0o|
|               S . +++oX|
|                . .. o*o|
|                 ....=|
|                o.o.o|
|                 .=00.|
+-----[SHA256]-----+
```

---

8. Confirm the generated content by running the `ls -la` command, as shown in Example 3-38.

*Example 3-38 Displaying the public and private RSA key pair*

---

```
# ls -la
total 52
drwxr-sr-x 2 qsecofr 0 12288 Sep 22 20:47 .
drwxr-sr-x 7 qsecofr 0 24576 Sep 22 17:31 ..
-rw----- 1 qsecofr 0 2602 Sep 22 20:47 id_rsa
-rw-r--r-- 1 qsecofr 0 565 Sep 22 20:47 id_rsa.pub
```

---

9. Before copying the SSH key to the managed hosts, install `sshpass`, which is a tool that facilitates password authentication in both interactive and non-interactive modes. Use the command in Example 3-39 to install `sshpass`.

*Example 3-39 Installing `sshpass` by using the `yum` command*

---

```
# yum install sshpass
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package sshpass.ppc64 0:1.06-1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch          Version
Repository              Size
=====
Installing:
 sshpass                ppc64        1.06-1
ibm                     30 k

Transaction Summary
=====
Install      1 Package

Total download size: 30 k
Installed size: 77 k
Is this ok [y/N]: Y
Downloading Packages:
sshpass-1.06-1.ibmi7.2.ppc64.rpm
| 30 kB 00:00:00
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : sshpass-1.06-1.ppc64
1/1

Installed:
  sshpass.ppc64 0:1.06-1

Complete!
```

---

### **IBM i hardware requirements for Ansible Controller**

The hardware resources that are required for Ansible Controller on IBM i depend on several factors, such as the number of agents that are served, frequency of operator check-ins, managed resources per agent, and the complexity of manifests and modules in use.

For an IBM i server with Ansible Controller installed, Table 3-4 on page 149 outlines best practice hardware specifications based on the number of managed nodes.

Table 3-4 Hardware requirements for Ansible Controller on IBM i

Managed nodes	CPU (cores)	Memory (GB)	Disk (GB)	IBM i release
Dozens	0.25	2	80	7.3, 7.4, and 7.5
1000+	0.5	8	80	7.3, 7.4, and 7.5

**Note:** These hardware requirements help ensure optimal performance and efficient management of Ansible tasks and operations on your IBM i server.

## 3.4 Preparing your systems to be Ansible clients

In general, there is minimal setup that is required to run the Ansible client on a device or server because Ansible does not require that an agent is installed on the client, and it uses SSH to connect the management node to the client node.

However, there are some basic setup considerations for each of the LPARs that are Ansible clients. The obvious one is to help ensure that SSH is installed and available. All supported OSs that run on IBM Power support SSH, but there are some considerations for helping ensure that it is installed correctly, which can vary by OS. Also, Ansible requires that Python is installed, and again this process differs depending on the OS that is used in your client.

The next sections describe the best practices to prepare your Ansible client based on the OS that is used.

### 3.4.1 Linux as an Ansible managed client

For a Linux node to work as an Ansible Client natively, you must install `python3` and `python3-pip`, and setup some parameters on the `ssh` configuration.

The following tasks help avoid delays that are caused by DNS resolution, or SSH timeouts:

1. In `/etc/ssh/sshd_config`, make the following changes:

```
Uncomment GSSAPIAuthentication no
Uncomment GSSAPICleanupCredentials yes
Uncomment UseDNS No
```

2. Add your `ansible-core` node to the `/etc/hosts` file if no DNS is setup.
3. Verify your `python3` installation, as shown in Example 3-40.

*Example 3-40 Verifying your Python installation*

---

```
[root@hugo-rhel8-ansible ~]# dnf list python3*
Updating Subscription Management repositories.
Installed Packages
python3-asn1crypto.noarch      0.24.0-3.e18      @anaconda
python3-cffi.ppc64le           1.11.5-5.e18      @anaconda
python3-configobj.noarch       5.0.6-11.e18      @anaconda
python3-cryptography.ppc64le  2.3-3.e18         @anaconda
```

---

4. Create a user for connection or set up `ssh` keys if you want to use passwordless access.

## 3.4.2 AIX as an Ansible managed client

This section describes tips and hints to use AIX as an Ansible managed client by using SSH or the AIX Collection from Ansible Galaxy.

To avoid delays that can be caused by DNS resolution or SSH timeouts, complete the following steps:

1. Modify `/etc/ssh/sshd_config` by changing the following lines:  
Uncomment `GSSAPIAuthentication no`  
Uncomment `GSSAPICleanupCredentials yes`  
Uncomment `UseDNS No`
2. Add your `ansible-core` node to the `/etc/hosts` file if no DNS is setup, and check `/etc/netsvc.conf`.
3. Create a user for connection (we used `ansible`) or set up SSH keys if you want to use passwordless access.

**Important:** From a security prospective, you should not use `root` for Ansible. As a best practice, create a separate user for Ansible. If you must escalate your privileges, you can use `su` with a password, `sudo`, or AIX native RBAC to achieve escalation. `Sudo` is available as a package from the AIX Toolbox for Open Source Software.

4. Verify your `python3` installation.

### Python installation considerations

Python is a primary prerequisite for your AIX Ansible client nodes. The method that you use to install Python depends on the version of AIX that you use.

First, verify whether `python3` is installed by using the `dnf` command, as shown in Example 3-41.

*Example 3-41 Verifying that `python3` is installed*

---

```
atilio-ansibleaix73: />dnf list python3*
Last metadata expiration check: 1 day, 3:34:59 ago on September 10, 2023 at
10:17:18 AM -03.
Installed Packages
python3.ppc                3.9.16-0      @System
python3-dnf.noarch        4.2.17-64_6   @System
python3-gpg.ppc           1.13.1-64_3   @System
python3-hawkey.ppc        0.39.1-64_5   @System
python3-libcomps.ppc      0.1.15-64_1   @System
python3-libdnf.ppc        0.39.1-64_5   @System
python3-librepo.ppc       1.11.0-64_2   @System
python3-pip.noarch        22.2.2-1      @AIX_Toolbox_noarch
python3.9.ppc             3.9.16-0      @System
```

---

Starting with AIX 7.3, Python is a standard part of the AIX distribution. You can check whether your specific AIX installation has Python that is installed by running the `ls1pp` command that is shown in Example 3-42 on page 151.

*Example 3-42 Checking the Python installation on AIX 7.3*

---

```
# lspp -L python3.9.base
```

File set	Level	State	Type	Description (Uninstaller)
python3.9.base	3.9.12.0	C	F	Python 3.9 64-bit binary distribution

---

On AIX versions earlier than 7.3, it is a best practice to use Python from the AIX Toolbox for Open Source Software. We described the process of DNF installation in 3.3.2, “AIX as an Ansible Controller” on page 134. Python is installed together with DNF.

Depending on how you installed Python, the main Python binary can be either `/opt/freeware/bin/python3` or `/usr/bin/python3`. For the sake of simplicity and standardization, choose one standard path to access `python3` across your whole environment. If you use `/usr/bin/python3`, you do not need to make any other changes in your Ansible playbooks, but you if you choose another location, then your future playbooks or inventories must define the variable `<ansible_python_interpreter>` with the full path to `python3` for your clients.

Wherever you install Python, be sure that you update your `PATH` settings. It is also a best practice that you create links to Python in `/usr/bin/`, as shown in Example 3-43.

*Example 3-43 Creating a link to Python in /usr/bin/*

---

```
atilio-ansibleaix73: />ln -s /usr/bin/python3 /usr/bin/python
ln: /usr/bin/python exists. Specify -f to remove.
atilio-ansibleaix73: />ls -lrt /usr/bin/python
lrwxrwxrwx  1 root  system          16 Sep 11 09:05AM /usr/bin/python ->
/usr/bin/python3
atilio-ansibleaix73: />ls -lrt /usr/bin/python3
lrwxrwxrwx  1 root  system          30 Sep 01 09:06AM /usr/bin/python3 ->
/usr/opt/python3/bin/python3.9
```

---

### 3.4.3 IBM i as an Ansible managed client

This section explores the convergence of IBM i with the Ansible ecosystem as a managed client. This integration empowers efficient system management and configuration by using Ansible's automation capabilities. You learn how IBM i becomes part of the Ansible managed environment, which enhances operational efficiency and configuration control.

## Enabling managed nodes on IBM i

The systems that can be managed by the Ansible Controller are known as IBM i endpoints or managed nodes. To prepare each endpoint system for management, the following requirements must be met:

1. Required License Program Products (LPPs):
  - a. IBM Portable Utilities for i (5733-SC1 option base)
  - b. OpenSSH, OpenSSL, and zlib functions (5733-SC1 option 1)
  - c. IBM HTTP Server for i (5770-DG1 option base)

**Note:** To determine whether these LPPs are already installed, you can use the following SQL queries from a 5250 terminal:

```
STRSQL
select * from QSYS2.SOFTWARE_PRODUCT_INFO where product_id = '5733SC1';
select * from QSYS2.SOFTWARE_PRODUCT_INFO where product_id = '5770DG1';
```

If they are not installed, you can download them from [Entitle Systems Support](#). For download and installation instructions, see [IBM Support](#).

2. To check the open-source packages and help ensure that Python 3.6+ is available, run the following commands:

```
yum search python | grep 3
python --version
```

**Note:** If these packages are not installed, install them by running the following command:

```
yum install python3 python3-itookit python3-ibm_db
```

3. To automatically start SSH after an initial program load (IPL), run the following command:

```
system "chgtcpsvr svrspcval(*sshd) autostart(*yes)"
```

4. Make sure that the home directory exists for the user that is defined as the Ansible user. To create a home directory on the IBM i LPAR to store the user's SSH-related objects, run the following command:

```
mkdir -p /home/<user>/.ssh,
```

The <user> variable must be changed to your user ID.

5. Transfer the generated public key (id\_rsa.pub) that is shown in Example 3-38 on page 147 from the Ansible Controller on IBM i to the managed nodes or endpoints. Example 3-44 illustrates the process.

*Example 3-44 Transferring the public key from the control node to the managed node*

```
# ssh-copy-id qsecofr@192.168.1.100
/QOpenSys/pkgs/bin/ssh-copy-id: INFO: Source of keys to be installed:
"/HOME/QSECOFR/.ssh/id_rsa.pub"
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
ECDSA key fingerprint is SHA256:Ino5PMGRhgupc23tpStiFRE4cPxVEmpZ/dmN1kfJ1RQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/QOpenSys/pkgs/bin/ssh-copy-id: INFO: attempting to log in with the new keys to
filter out any that are already installed
/QOpenSys/pkgs/bin/ssh-copy-id: INFO: 1 key remains to be installed -- if you are
prompted now it is to install the new keys
qsecofr@192.168.1.100's password: "type the password"
sh: test: argument expected
```



Number of key(s) added: 1

Now, try logging in to the machine with: "ssh 'qsecofr@192.168.1.100'" and check to make sure that only the keys you wanted were added.

---

6. Help ensure that the authorized SSH key is successfully transferred to the managed node. To confirm, perform a passwordless login to the managed host, as shown in Example 3-45.

*Example 3-45 Passwordless login to the managed host*

---

```
qsecofr@CONTROLLER:~# ssh qsecofr@192.168.1.100
```

```
GNU bash, version 4.4.23(1)-release (powerpc-ibm-os400)
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

```
This shell has been enhanced by .dotfiles version 1.3.0
```

```
qsecofr@MANAGED:~#
```

---

## Using the IBM i Ansible collection

“Enabling managed nodes on IBM i” on page 152 provided information about how to enable an IBM i server as a managed node or client in your Ansible management environment. This section describes some extra steps to take and provides best practices to make the management of your IBM i clients more effective.

### ***Building an Ansible Inventory***

Creating an accurate inventory is fundamental for Ansible's effective management of different managed nodes or endpoints. To build a well-structured Ansible inventory, follow these guidelines:

- ▶ **Managed nodes:** Ansible Controller interacts with various managed nodes or endpoints, each serving a specific purpose in your infrastructure.
- ▶ **Default inventory location:** The default inventory file is named `hosts`, and its location is typically `/etc/ansible/hosts`.
- ▶ **Custom inventory path:** You have the flexibility to specify an alternative inventory file path by using the `-i <path>` option while running commands.
- ▶ **Multiple inventory files:** Ansible supports using multiple inventory files simultaneously, which offers adaptability for complex environments.
- ▶ **Creating an inventory file:** To create your own inventory file, run the following command:  

```
vi /etc/ansible/hosts
```

Example 3-46 shows the content of an inventory file

*Example 3-46 Inventory file for IBM i demonstration*

---

```
[local]
localhost ansible_connection=local
[ibmi]
192.168.1.100 ansible_ssh_user=avalosm ansible_ssh_pass=abc123
[ibmi:vars]
ansible_python_interpreter="/Q0pensys/pkgs/bin/python3"
```

---

- ▶ **IBM i Collections inventory:** The Ansible Galaxy IBM i collections offer a preexisting inventory file at the following location:

`~/ansible/collections/ansible_collections/ibm/power_ibmi/playbooks/`

**Note:** This inventory can be edited to fit your needs. You can the `hosts_ibmi.ini` file in this directory for this purpose.

- ▶ **Check Inventory configuration:** To verify your current inventory configuration, use the following command:

```
ansible-inventory --list -y
```

**Tip:** Building a comprehensive inventory helps ensure that Ansible can effectively manage the specified nodes and endpoints. This foundation is essential for orchestrating automation and configuration tasks across your infrastructure.

**Best practice:** For demonstration and general understanding in this chapter, we used the QSECOFR user profile. However, it is a best practice to NOT use the QSECOFR user profile with SSH on IBM i. This best practice applies to both the Ansible Controller and the managed nodes or endpoints. Create a user profile with comparable authority levels.

For this new user profile, create a HOME directory on the IBM i system. Also, configure appropriate permissions for the user's profile HOME directory by running the `chmod 755` command. Modify the ownership of the HOME directory to match the SSH user. Integrate the HOME directory into the user profile.

In alignment with Ansible's prerequisites for generating and copying public keys, it is crucial to establish a dedicated directory within the user's HOME directory that is named `ssh`. Configure permissions for this SSH directory by using the `chmod 700` command to help ensure the appropriate level of security.

### ***Ansible ad hoc commands***

Ansible ad hoc commands provide an approach to running essential operations on the IBM i server. Operating from the Ansible Controller, these commands enable specific tasks to run efficiently. Each command focuses on a single operation for quick task execution. If multiple tasks are required, use a sequence of commands. For added flexibility, specify alternative inventory file locations by using the `-i [inventory]` option.

The Ansible Controller or control node itself can be incorporated as a managed host within the inventory file. This function enhances the efficiency of running routine tasks.

The following examples demonstrate the practical applications of Ansible ad hoc commands:

- ▶ To verify the readiness of all inventory hosts for management by the Ansible Controller, run the command that is shown in Example 3-47.

*Example 3-47 Verifying host availability with an Ansible ad hoc command*

---

```
# ansible all -m ping
```

---

- ▶ To vary an IBM i device, run the command that is shown in Example 3-48.

*Example 3-48 Varying IBM i devices by using an Ansible ad hoc command*

---

```
# ansible ibmi -m ibm_power_ibmi.ibm_i_device_vary -a 'device_list=OPT02,status=*ON
joblog=false'
```

---

### **Ad hoc commands and playbooks for IBM i**

Ad hoc commands are useful for straightforward tasks, such as variably activating or deactivating resources on IBM i, and testing the reachability of managed hosts. In this process, the Ansible Controller dispatches a Python script to the managed nodes. The script reports back the success or failure of the operation. Ad hoc commands trigger only one module and its corresponding set of arguments at a time.

Playbooks are ideal for more intricate configurations or orchestration scenarios, often involving a series of tasks that collectively run a larger action by using modules. In the context of IBM i modules, common core modules such as `find` are supported. Playbooks are well suited for various DevOps practices. Some typical scenarios where playbooks excel include IBM i configuration management, orchestrating IBM i deployments, and managing tasks after deployments take place.

### **Crafting effective YAML playbooks for IBM i**

Using Ansible playbooks introduces a powerful approach to orchestrate a multitude of tasks across multiple IBM i systems. These playbooks enable execution of various actions, such as helping ensure consistent configurations, performing common tasks, running deployments, and more. Expressed in YAML syntax, playbooks provide a structured framework for orchestrating tasks, enhancing efficiency, and promoting system consistency.

Here are some functions of playbooks:

- ▶ Incorporate comments: To enhance readability and comprehension, you can integrate comments that use “#” symbol that is integrated into your YAML playbook. These comments provide contextual insights and explanations for the included tasks or configurations.
- ▶ Playbook initialization: YAML playbooks commence with the declaration of three hyphens (---) that initiate the definition of the playbook's structure and content, which creates a clear demarcation for later actions.
- ▶ Specifying host inventory: The playbook designates the host inventory that is engaged in the orchestrated tasks. This specification helps ensure that the defined tasks are accurately targeted toward the intended systems.
- ▶ Gathering facts: For efficient playbook execution, disable unnecessary fact gathering from managed nodes. In Ansible 2.8 and later, the directive “gather\_facts: no” is employed to curtail fact collection.
- ▶ Harnessing IBM i collections: IBM i playbooks benefit from dedicated collections that encapsulate tailored modules and functions. By including the relevant collection (for example, “`ibm.power_ibmi`”) in the playbook, administrators can seamlessly access specific IBM i capabilities.

- ▶ **Defining tasks:** The heart of the playbook lies within the “tasks” section, where the orchestrated operations are defined. Each task encompasses the following items:
  - A descriptive name (“name”) that describes the purpose of the task.
  - The module that performs the action.
  - Corresponding arguments that tailor the task’s behavior.
- ▶ **Multiplexing tasks:** The playbook accommodates the definition of multiple tasks to enable the execution of a series of actions in a coherent sequence. This versatility empowers administrators to create comprehensive automation scenarios.
- ▶ **Concluding the playbook:** Conclude the playbook by using three dots (“...”), which signify the end of the defined content. This conclusion is optional, but contributes to maintaining a playbook.

Example 3-49 provides a comprehensive playbook sample for IBM i that showcases various YAML syntax features.

*Example 3-49 Sample playbook for IBM i*

---

```
# Sample Playbook for IBM i
---
hosts: ibmi
gather_facts: no
collections:
  - ibm.power_ibmi
tasks:
  - name: Display a system value
    ibmi_sysval:
      sysvalue:
        - {'name': 'qccsid'}
    register: dspsysval_ccsid_result

  - name: Display the returned parameters
    debug:
      msg: "{{ chksysval_qmaxsign_result }}"
...

```

---

**Note:** By following these guidelines and crafting YAML playbooks, administrators unlock Ansible's full potential for IBM i. This approach optimizes automation efficacy, which helps ensure consistency and reliability across IBM i systems. For more information about YAML syntax, see the comprehensive YAML documentation at the [YAML site](#).

## ***Idempotency and its significance on IBM i***

Idempotency serves as a characteristic that amplifies the value of running Ansible playbooks on IBM i environments. Understanding its implications can enhance the predictability and stability of your operations:

- ▶ **Reliable execution with consistent results:** Idempotency is a remarkable attribute of Ansible playbooks that enables them to run multiple times without altering or disrupting existing configurations. This unique quality helps ensure that when a playbook is run repeatedly, it consistently produces the same outcomes, which contribute to the reliability and predictability of your automation processes.
- ▶ **Intrinsic to modules:** The concept of idempotence is intrinsic to Ansible modules, and forms a fundamental pillar of their design philosophy. Ansible modules aim to achieve consistent results regardless of how many times they are started, which reinforces the reliability of your automation tasks.
- ▶ **Idempotency on IBM i:** Ansible modules on IBM i are inherently idempotent by default. This foundational quality helps ensure that the operations that are performed by these modules adhere to the principles of idempotency, which simplifies management and minimizes the risk of unintended alterations.

Here is a practical example. Consider the IBM i module “cmd” that is sourced from the official repository ([ibmi\\_cl\\_command.rst](#)), from which you call the following program:

```
CALL QZLSMAINT PARM('40' '1' '0x100')
```

In this case, the configuration flags are cumulative, where repeated executions can add 0x100 to the value. This situation can potentially disrupt the NetServer configuration. There is no inherent safeguard to prevent administrators from unintentionally affecting the NetServer configuration through repeated calls to QZLSMAINT.

**Note:** You can help ensure consistent and dependable automation outcomes by grasping the principle of idempotency and its implementation within Ansible modules on IBM i. This awareness permits administrators to use automation confidently while safeguarding against unintended consequences, which helps ensure the stability and integrity of your systems.

## ***Using the power\_ibmi modules***

When you work within the IBM i environment, You can use a comprehensive set of Ansible modules to streamline administrative and deployment tasks. These modules serve as fundamental building blocks to enable interaction with IBM i servers. With a diverse array of modules that are available, each module is meticulously crafted to perform specific tasks, which effectively cover a wide spectrum of administrative and deployment needs.

To harness the power of these modules, you must understand their parameters and data types. Each module is accompanied by its own set of parameters, each tailored to cater to distinct functions. These parameters dictate the behavior and configuration of the module during execution. Whether you are creating, modifying, or managing resources on the IBM i, mastering the usage of these resources and modules, referring to the official documentation is crucial. This documentation, accessible at [IBM i modules](#), provides in-depth insights into each module's capabilities, parameter details, and usage examples. By exploring the documentation, IBM i administrators and developers can unlock the full potential of Ansible's power\_ibmi modules, making way for smoother, more efficient administration and deployment processes.

### ***Running precise CLI commands by using the `ibmi_cl_command` module***

The `ibmi_cl_command` module plays a crucial role in the Ansible toolkit for IBM i servers by providing administrators and operators with a direct method to run CLI commands. This module's core function is centered on the `cmd` parameter, which serves as the conduit for passing a specific CLI command for execution.

However, the `ibmi_cl_command` module has specific boundaries. Unlike a conventional 5250 emulator, this module does not facilitate interaction with menus or commands in the same way. Instead, its primary purpose is to efficiently run CLI commands in a controlled and automated manner.

A significant feature of this module is its adaptability regarding user context. The module can be configured to run either as the user that establishes the SSH connection, usually an administrator, or as a designated user that uses the “`become_user`” and “`become_user_password`” parameters. This adaptability helps ensure that tasks run within the correct user context, which accommodates various operational scenarios.

By mastering the `ibmi_cl_command` module, administrators and operators can harness a powerful tool to manage and automate tasks within the IBM i environment. An exploration of the module's practical applications and real-world instances can provide valuable insights into effectively using its capabilities. Example 3-50 shows a sample of the module.

#### *Example 3-50 Sample of the `ibmi_cl_command` module*

---

```
- hosts: ibmi
  gather_facts: false
  collections:
    - ibm.power_ibmi
  tasks:
    - name: Display the user profile
      ibmi_cl_command:
        cmd: dspusrprf usrprf (AVALOSM)
        output: "*PRINT"
        register: dspusrprf_result

    - name: Print the user profile stdout lines
      debug:
        msg: "{{ dspusrprf_result.stdout_lines }}"
```

---

### ***Discovering IBM i objects with precision by using `object_find`***

The `object_find` module emerges as a valuable asset within the Ansible array of tools for IBM i systems by offering an approach to discovering specific IBM i objects based on user-defined criteria. This module operates as a versatile search mechanism that can find objects by employing various criteria that can be combined by using logical “AND” operators.

A stand-out feature of the `object_find` module is its ability to conduct searches by using a comprehensive range of parameters. These parameters permit attributes such as object age, size, name, type, and library, among others. This versatility grants administrators and operators the means to precisely pinpoint objects within the IBM i environment, which caters to a multitude of scenarios and requirements.

Behind the scenes, the `object_find` module relies on the integration of the IBM i `QSYS2.OBJECT_STATISTICS` and `QSYS2.SYS_SCHEMAS` views. This integration forms the backbone of the module's efficiency so that it can swiftly retrieve pertinent information and present it in a coherent and actionable manner.

By using the `object_find` module, administrators and operators can swiftly navigate and extract valuable insights from their IBM i servers. The module's ability to run nuanced searches enhances the management and automation of tasks, which provide a versatile solution to address diverse operational needs.

Example 3-51 shows an example of the module that is described on this section.

*Example 3-51 Sample of `object_find` module*

---

```
- hosts: ibmi
gather_facts: false
collections:
  - ibm.power_ibmi
tasks:
  - name: Find all journals and journal receivers in library AVALOSM
    ibm.power_ibmi.ibm_object_find:
      object_name: '*ALL'
      object_type_list: '*JRN *JRRCV'
      lib_name: 'AVALOSM'
      age: '1w'
      age_stamp: 'ctime'
      register: journals

  - name: Print the user profile stdout lines
    debug:
      msg: "{{ journals }}"
```

---

### ***Extracting insights with SQL precision by using `ibmi_sql_query`***

The `ibmi_sql_query` module stands as a potent tool within the Ansible toolkit that permits administrators and operators to harness the power of SQL to retrieve valuable information from Db2 for IBM i. Operating as a bridge to the extensive wealth of system information that is held within Db2, this module streamlines the process of querying data, which makes it an indispensable asset for effective system management.

One of the core strengths of the `ibmi_sql_query` module is its seamless execution of SQL queries. By interfacing with Db2 for i, users can tap into a treasure trove of insights that are housed within tables, views, and various SQL services. These insights span crucial aspects of system functioning, offering a comprehensive view of system health, resource allocation, and performance metrics.

Furthermore, the `ibmi_sql_query` module extends its functions with the `expected_row_count` parameter. With this added feature, administrators can fine-tune their querying tasks that enables them to define expectations for query results. If the result set does not meet the defined expectations, the module can be configured to trigger a task failure, which provides an automated quality assurance (QA) mechanism.

By integrating the capabilities of SQL querying into Ansible workflows, the `ibmi_sql_query` module bolsters the toolkit for IBM i administrators and operators. This module's ability to explore into the inner workings of Db2 for i enhances the precision of system monitoring, diagnostics, and optimization. As a result, Ansible users can navigate the intricacies of their IBM i environments with greater efficiency and depth of insight.

Example 3-52 displays a sample that uses the SQL query module.

*Example 3-52 Sample for SQL query*

---

```
- hosts: ibmi
gather_facts: false
collections:
  - ibm.power_ibmi
tasks:
  - name: check if the user exists
    ibm.power_ibmi.ibm_sql_query:
      sql: "SELECT * FROM QSYS2.USER_INFO_BASIC WHERE AUTHORIZATION_NAME =
'AVALOSM'"
    register: user_query_result

  - name: display the result from the query
    debug:
      msg: "{{ user_query_result }}"

  - name: assert the user doesn't exist
    assert:
      that: (user_query_result.row | length) == 0
```

---

### 3.4.4 Virtual I/O Server as an Ansible managed client

Businesses are turning to PowerVM virtualization to consolidate multiple workloads onto fewer systems to increase server usage and reduce cost. PowerVM provides a secure and scalable virtualization environment for AIX, Linux, and IBM i applications that are built on the advanced reliability, availability, and serviceability (RAS) features and the leading performance of the IBM Power platform.

The Virtual I/O Server (VIOS) is part of the PowerVM hardware feature. The VIOS is software that is in an LPAR that runs in your IBM Power server that provides virtualization functions for the other LPARs that run in that server. The VIOS provides virtual SCSI target, virtual Fibre Channel, Shared Ethernet Adapter, and PowerVM Active Memory Sharing capabilities to client LPARs within the system. The VIOS also provides the Suspend/Resume feature to AIX, IBM i, and Linux client LPARs within the system. You can use the VIOS to perform the following functions:

- ▶ Sharing of physical resources between LPARs on the system
- ▶ Creating LPARs without requiring extra physical I/O resources
- ▶ Creating more LPARs than there are I/O slots or physical devices that are available, with the ability for LPARs to use dedicated I/O, virtual I/O, or both
- ▶ Maximizing the usage of physical resources on the system
- ▶ Helping to reduce the storage area network (SAN) infrastructure

The VIOS is configured and managed through a CLI. All aspects of VIOS administration can be accomplished through the CLI, including the following items:

- ▶ Device management (physical, virtual, and logical volume manager (LVM))
- ▶ Network configuration
- ▶ Software installation and update
- ▶ Security
- ▶ User management
- ▶ Maintenance tasks



## Setting up your VIOS for Ansible client

The default user for connecting to a VIOS LPAR is padmin, which has access to the ioscli shell. In our example, we chose to create a unique user for our Ansible management environment, user ansible, as shown in Example 3-53.

### *Example 3-53 Creating the ansible user ID in the VIOS*

---

```
Last unsuccessful login: Wed Nov 10 12:23:52 CST 2021 on ssh from sltsmnim
Last login: Wed May 24 07:42:54 CDT 2023 on /dev/pts/1 from 192.168.184.201
The most recent software update has modified the current system rules.
These modifications have not been deployed on the system. To view the
modifications and deploy, run the 'rulescfgset' command.
```

```
$ oem_setup_env
# mkuser roles=PAdmin,CacheAdm,FSAdmin,pkgadm \
    default_roles=PAdmin,CacheAdm,FSAdmin,pkgadm ansible>
# id ansible
uid=205(ansible) gid=1(staff)
# pwdadm -c ansible
```

---

Starting with VIOS 4.1.0.10, Python is included in the VIOS image. If you use VIOS earlier versions, you must set up Python.

Example 3-54 shows that a simple Ansible command is unsuccessful because Python is not natively installed on the VIOS before Version 4.1.0.10.

### *Example 3-54 Ansible command failure due to Ansible not being set up*

---

```
[root@ansible-AAP-redbook playbooks]# cat facts.yml
---
- hosts: all
  remote_user: ansible

  tasks:
  - name: print facts
    debug:
      var: ansible_facts
[root@ansible-AAP-redbook playbooks]# ansible-playbook -i vios_inventory.yml
facts.yml --ask-pass
SSH password:

PLAY [all] *****

TASK [Gathering Facts] *****
fatal: [vio]: FAILED! => {"ansible_facts": {}, "changed": false, "failed_modules":
{"ansible_legacy_setup": {"failed": true, "module_stderr": "Shared connection to
bergessioviosl closed.\r\n", "module_stdout": "/bin/sh: /usr/bin/python: not
found.\r\n", "msg": "MODULE FAILURE\nSee stdout/stderr for the exact error", "rc":
127}}, "msg": "The following modules failed to run: ansible_legacy_setup\n"}

PLAY RECAP *****
vio                : ok=0    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0

[root@ansible-AAP-redbook playbooks]
```

---

You can use either of the following approaches:

- ▶ Install dnf from the AIX Toolbox.
- ▶ Install dnf by using the power of Ansible to run the installation

In this example, we opted to use option 2, so we created the playbook `dnf-vios.yml` to do the installation, as shown in Example 3-55.

*Example 3-55 Setting up su for ansible user, and vios-dnf.yml*

---

```
$ oem_setup_env
# passwd root
Changing the password for "root"
root's New password:
Enter the new password again:
# pwdadm -c root
# chuser su=true root

# cat -n dnf-vios.yml

- hosts: all
  remote_user: ansible
  gather_facts: no
  become: true
  become_method: su

  tasks:
    - name: download dnf_aixtoolbox.sh
      get_url:
        url:https:
//public.dhe.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/dnf_aixtoolbox.sh
        dest: /tmp/dnf_aixtoolbox.sh
        mode: 0755
        delegate_to: localhost
    - name: copy dnf_aixtoolbox.sh to vios
      raw: "scp -p /tmp/dnf_aixtoolbox.sh {{ ansible_user }}@{{ inventory_hostname
}}:/tmp/dnf_aixtoolbox.sh
        delegate_to: localhost
    - name: execute dnf_aixtoolbox.sh on vios
      raw: "chmod 755 /tmp/dnf_aixtoolbox.sh ; /tmp/dnf_aixtoolbox.sh -y"

# ansible-playbook -K -i viol, dnf-vios.yml
BECOME password:

PLAY [all]
*****
*****

TASK [download dnf_aixtoolbox.sh]
*****
ok: [viol - localhost]

TASK [copy dnf_aixtoolbox.sh to vios]
*****
```

```

changed: [viol - localhost]
*****

TASK [update virtual RPM packages]
*****

changed: [viol]

TASK [execute dnf_aixtoolbox.sh on
viol]*****

changed: [viol]

PLAY RECAP
*****
*****

viol          :oks4  changed=3  unreachable=@  failed=0  skipped=0  rescued=0
ignored=0

```

---

Now, we have dnf installed in our VIOS.

If you plan to have an HA environment, then there will be multiple VIOS in your environment because it is a best practice at least two VIOSs per-server frame, which acts as an active/active failover solution. If you have more requirements for the separation of environments for security, there might be more VIOSs on your server, and there almost certainly will be multiple servers in your environment.

You need to be sure that you can access all the VIOS LPARs. You can manually create the user on each VIOS, or use Ansible to create the users. In this example, we use Ansible to create a shell script for this purpose, as shown in Example 3-56.

*Example 3-56 Script to create users on VIOSs*

---

```

#!/usr/bin/expect -f

if { $argc != 3 } {
send_error "Usage : $argv@ ssh-connection ssh-password user-password\n"
exit 1
}

log_user 1

set timeout 60

set sshconn [lindex $argv 0]
set sshpw [lindex $argv 1]
set newpw [Llindex $argv 2]

spawn ssh $sshconn
expect "password: "
send "$sshpw\r"

expect "$ "

```

```

send "oem_setup_env\r"

expect "# "

send "mkuser roles=PAdmin,CacheAdm,FSAdmin,pkgadm,SysBoot,isso
default_roles=PAdmin,CacheAdm,FSAdmin,pkgadm,SysBoot,isso ansible\r"
expect "# "

send "chuser su=true root\r"

expect "# "

send "echo 'ansible:$newpw' | chpasswd -c\r"
expect "# "

send "echo 'root:$newpw' | chpasswd -c\r"
expect "# "

send "exit\r"
send "exit\r"

```

---

Example 3-56 on page 163 is used as part of our playbook to run user creation on every VIOS.

We have the script, so now we build an inventory to run that playbook, as shown in Example 3-57.

*Example 3-57 Inventory build for the Ansible user creation playbook*

---

```

#cat viosinventory
[vios]
vios1
vios2
vios3
vios4
[all:vars]
ansible_connection=ssh
ansible_user=padmin
ansible_password=padminpassword

```

---

The playbook that we used to create the ansible user on each VIOS is shown in Example 3-58.

*Example 3-58 Playbook create-vios-user.yml*

---

```

#cat create-vios-user.yml
---
- name: create remote user
  raw: "{{ download_dir }}/user.e {{ ansible_user }}@{{ inventory_hostname }} {{
ansible_password }} {{ new_password }}"
  delegate_to: localhost

- name: copy ssh public key
  raw: "SSHPASS={{ new_password }} sshpass -e ssh-copy-id ansible@{{
inventory_hostname }}"

```

```
delegate_to: localhost
```

```
#ansible-playbook -i vios-inventory create-vios-user.yml
```

---

Now that we have our users and inventory, we can install Python on all the partitions by using with a similar script than the one we used for creating the users. This script is shown in Example 3-59.

*Example 3-59 Installing the python3 and pip3 script*

---

```
#!/usr/bin/expect -f

if { $argc != 3 } {
send_error "Usage : $argv@ ssh-connection ssh-password user-password\n"
exit 1
}

log_user 1

set timeout 60

set sshconn [lindex $argv 0]
set sshpw [lindex $argv 1]

spawn ssh $sshconn
expect "password: "
send "$sshpw\r"

expect "$ "

send "oem_setup_env\r"

expect "# "

send "dnf install python3 pip3 -y \r"
expect "# "
```

---

Now, that our VIOS clients are ready for use, we walk through a set of use cases for the VIOS collection:

- ▶ VIOS upgrade
- ▶ VIOS backup
- ▶ VIOS mapping
- ▶ VIOS hardening

Validate that the collection is installed and ready. Example 3-60 validates the versions of the collections that are installed.

*Example 3-60 Validating the ibm.power collections*

---

```
root@ansible-AAP-redbook ~]# ansible-galaxy collection list
[WARNING]: Collection at '/root/.ansible/collections/ansible_collections/ibm/aix'
does not have a
MANIFEST.json file, nor has it galaxy.yml: cannot detect version.

# /root/.ansible/collections/ansible_collections
Collection      Version
-----
community.general 6.6.0
ibm.aix          *
ibm.power_aix    1.6.4
ibm.power_hmc    1.8.0
ibm.power_vios   1.2.3
[root@ansible-AAP-redbook ~]#
```

---

### **VIOS upgrade**

With our collections installed and ready, we start using them to write some powerful Ansible scripts for our servers. Example 3-61 shows a playbook that is used to update the VIOS software.

*Example 3-61 The vios-update.yml playbook*

---

```
#cat vios-update.yml

- hosts: all
  remote_user: ansible
  gather_facts: no
  collections:
  - ibm.power_aix
  - ibm.power_vios

  roles:

  - name: Bootstrap VIOS
    role: bootstrap_vios
    become: true
    become_method: su

  - name: Update VIOS (1st part)
    role: update_vios
    when: "'v1' in inventory_hostname"

  - name: Update VIOS (2nd part)
    role: update_vios
    when: "'v2' in inventory_hostname"
```

---

The roles are the most important part of this playbook. Example 3-62 on page 167 shows details about the update\_vios role.

*Example 3-62 Displaying the role update\_vios*

---

```
#cat roles/update_vios/tasks/main.yml
- name: commit all uncommitted updates
  ibm.power_vios.updateios:
    action: commit
- name: mount remote repository with updates
  ibm.power_aix.mount:
    state: mount
    node: "{{ repo_node }}"
    mount_dir: "{{ repo_dir }}"
    mount_over_dir: "{{ local_dir }}"
- name: update VIOS
  ibm.power_vios.updateios:
    action: update
    device: "{{ local_dir }}"
    accept_licenses: yes
```

---

However, the `vios-update` playbook that is shown in Example 3-61 on page 166 is not complete because it should address two more issues:

- ▶ Commit can fail with RC = 19 or 20, which means “Already committed”.
- ▶ After the update, restart your VIOS.

You can manually run the `updateios` playbook on the CLI and ignore some return codes, as shown in Example 3-63.

*Example 3-63 Code to commit and ignore return codes 19 and 20*

---

```
---
- name: commit all uncommitted updates
  shell:
  cmd: fusr/ios/cli/ioscli updateios -commit
  register: result
  failed_when: ( result.rc not in [ 0, 19, 20 ] )
```

---

### **Backing up VIOS**

Now, look at some playbooks to back up your VIOS. These playbooks run `viosbr`, which is an `mksysb` for VIOSs. This Ansible playbook that is shown in Example 3-64 runs `viosbr` and directs the output to an NFS share location.

*Example 3-64 Playbook to run viosbr*

---

```
#cat viosbr-playbook.yml
---
- name: backup vios
  hosts: all
  gather_facts: false
  remote_user: ansible
  vars:
    ansible_python_interpreter: /opt/freeware/bin/python3

  tasks:
  - name: get current date
    ansible.builtin.shell: "date +%Y%m%d"
    register: ourdate
    delegate_to: localhost
```

```

- name: create viosbr backup
  ibm.power_vios.vioshr:
    action: backup
    file: "/home/ansible/vioshr.{{ ourdate.stdout }}"

- name: mount NFS share for backups
  ibm.power_aix.mount:
    state: mount
    node: nim
    mount_dir: /backup
    mount_over_dir: /mnt

- name: create mksysb of VIO
  ibm.power_vios.backupios:
    file: "/mnt/{{ inventory_hostname }}_mksysb.{{ ourdate.stdout }}"
    mksysb: true

- name: unmount NFS share
  ibm.power_aix.mount:
    state: umount
    mount_over_dir: /mnt

```

---

### ***VIOS mapping***

You can use an Ansible playbook to discover facts about your VIOSs. Example 3-65 shows a script that you can use for discovery.

*Example 3-65 The vios-facts.yml script*

---

```

#cat vios-facts.yml
---
- name: get VIO mapping
  hosts: all
  remote_user: ansible
  gather_facts: false
  vars:
    ansible_python_interpreter: /opt/freeware/bin/python3

  tasks:
    - name: get mapping facts
      ibm.power_vios.mapping_facts:
    - name: print facts
      ansible.builtin.debug:
        var: ansible_facts

```

---

### ***VIOS hardening***

Example 3-66 is a script that sets the security level for the VIOS.

*Example 3-66 The vios-security.yml script*

---

```

#cat vios-security.yml
---
- name: apply secure configuration
  ibm.power_vios.viosecur:
    level: low

```

---



### 3.4.5 Red Hat OpenShift as an Ansible managed client

Red Hat OpenShift on IBM Power can run in many different environments. For example, you can use a bare metal installation on your Power server or you can use LPARs, which have are prebuilt by using PowerVM. These two scenarios are mostly manual and they do not gain much advantage from the usage of Ansible (although you might have used Ansible to build the LPARs).

Ansible is more help if you are deploying your Red Hat OpenShift cluster on Power Systems Virtual Server, on PowerVM that is managed by PowerVC, or on Power servers that are managed by KVM. These scenarios are described in this section.

The environment is shown in Figure 3-22.

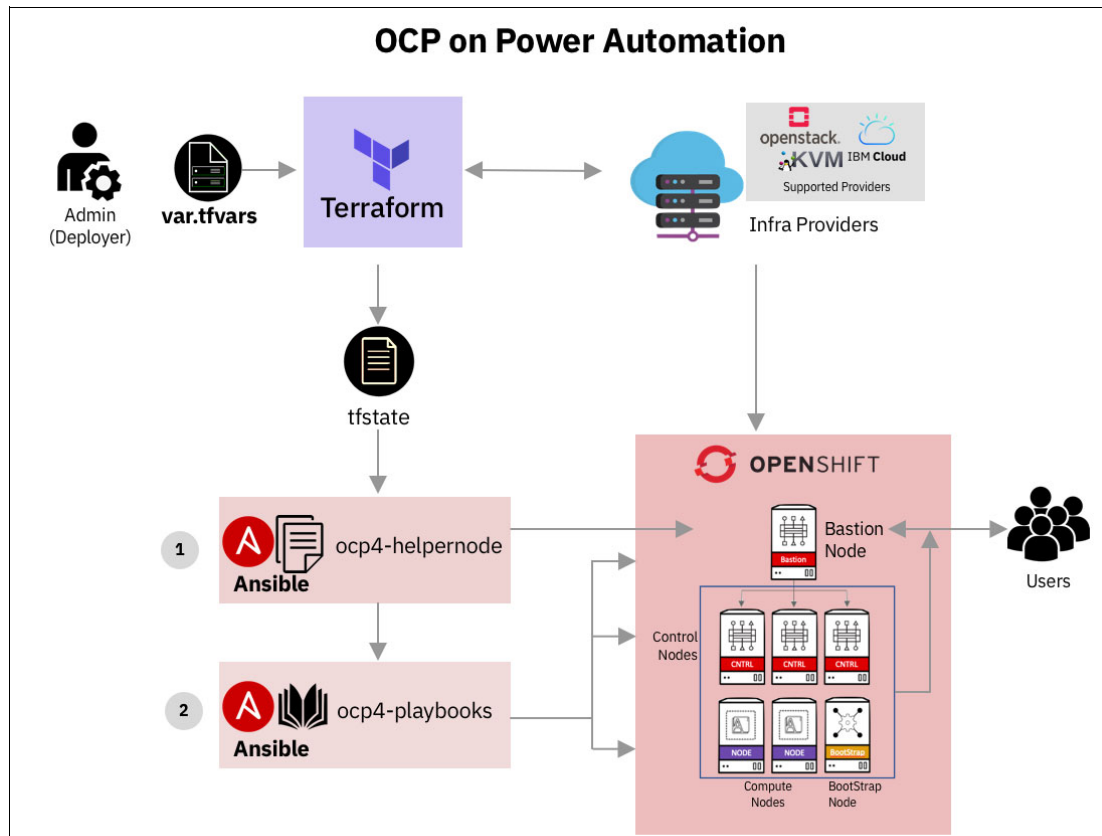


Figure 3-22 Red Hat OpenShift deploy with Ansible scenarios

### Installing Red Hat OpenShift on Power Systems Virtual Server by using Ansible

Power Systems Virtual Server features a CLI for managing and creating your servers, In this section, we show you scripts that use Terraform and Ansible to deploy a Red Hat OpenShift cluster on Power Systems Virtual Server by using ocp4-upi-powervs for the deployment process.

You must have an IBM Cloud account to deploy Power Systems Virtual Server resources. To help ensure that your Power Systems Virtual Server instance can deploy Red Hat OpenShift clusters, make sure that your account has the proper permissions and validate that you have created the appropriate security certificates.

### **Detailed prerequisites**

This section helps ensure that your IBM Cloud account is set up and that you can create the Power Systems Virtual Server instance for your Red Hat OpenShift cluster. To set up your environment, complete the following steps:

1. Validate that you have an IBM Cloud account to create your Power Systems Virtual Server Instance (VSI). If you do not have an account, create an account at [IBM Cloud](#).
2. Create an IBM Cloud account API key. For more information about setting up your API key, see *IBM Power Virtual Server Guide for IBM AIX and Linux*, SG24-8512 or the [IBM Cloud documentation](#).
3. After you have an active IBM Cloud account, you can create a Power Systems Virtual Server service. For more information, see *IBM Power Virtual Server Guide for IBM AIX and Linux*, SG24-8512.
4. Next, request an Red Hat OpenShift Pull secret, which you can download from [Red Hat IDP](#). Place the file into the installation directory, and name it `pull-secret.txt`. You need an RHEL subscription ID and password.

### **Getting ready for installation**

The installation uses a simple script-based installer that deploys an Red Hat OpenShift cluster on Power Systems Virtual Server. The script can run in multiple platforms, including Linux (x86\_64/ppc64le), Windows, and Mac OSX.

The Power Systems Virtual Server instance requires special network permissions to ensure that inbound access is allowed for TCP ports for ssh (22) and allow outbound access for http (80), https (443), and OC CLI (6443). These network permissions are required only when using a cloud instance or a remote VM so that you can connect to it by using SSH and run the installer.

To prepare for the installation, complete the following steps:

1. Create an install directory where all the configurations, logs, and data files are stored:  

```
[root@ansible-AAP-redbook ~]# mkdir ocp-install-dir && cd ocp-install-dir
```
2. Download the script on your system and change the permission to run, as shown in Example 3-67.

#### *Example 3-67 Changing the script to enable the run time*

---

```
[root@ansible-AAP-redbook ocp-install-dir]# curl -sL
https://raw.githubusercontent.com/ocp-power-automation/openshift-install-power/mai
n/openshift-install-powervs -o ./openshift-install-powervs
[root@ansible-AAP-redbook ocp-install-dir]# ls -lrt
total 64
-rw-r--r--. 1 root root 62001 Sep 29 15:24 openshift-install-powervs
[root@ansible-AAP-redbook ocp-install-dir]# chmod +x ./openshift-install-powervs
[root@ansible-AAP-redbook ocp-install-dir]#
```

---

Running the script without parameters displays the help information for the script, as shown in Example 3-68.

#### *Example 3-68 Help information for the script*

---

```
[root@ansible-AAP-redbook ocp-install-dir]# ./openshift-install-powervs
```

Automation for deploying Red Hat OpenShift 4.X on Power Systems Virtual Server

Usage:

```
openshift-install-powervs [command] [<args> [<value>]]
```

Available commands:

```
setup          Install all the required packages/binary files in the current
               directory
variables      Interactive way to populate the variables file
create        Create an Red Hat OpenShift cluster
destroy       Destroy an Red Hat OpenShift cluster
output        Display the cluster information. Runs terraform output [NAME]
access-info   Display the access information of installed Red Hat OpenShift
cluster
help          Display this information
```

Where <args>:

```
-var           Terraform variable to be passed to the create/destroy command
-var-file     Terraform variable file name in current directory. (By
default using var.tfvars)
-flavor       Cluster compute template to use for example: small, medium,
               large
-force-destroy Not ask for confirmation during destroy command
-ignore-os-checks Ignore operating system-related checks
-ignore-warnings Warning messages will not be displayed. Should be specified
               first before any other args.
-verbose      Enable verbose for terraform console messages
-all-images   List all the images available during the variables prompt
-trace        Enable tracing of all ran commands
-version, -v  Display the script version
```

Environment Variables:

```
IBMCLOUD_API_KEY  IBM Cloud API key
RELEASE_VER       Red Hat OpenShift release version (Default: 4.13)
ARTIFACTS_VERSION Tag or Branch name of ocp4-upi-powervs repository (Default:
main)
RHEL_SUBS_PASSWORD RHEL subscription password if not provided in variables
NO_OF_RETRY       Number of retries/attempts to run repeatable actions such as
create (Default: 5)
```

Submit issues at:

```
https://github.com/ocp-power-automation/openshift-install-power/issues
```

```
[root@ansible-AAP-redbook ocp-install-dir]#
```

---

3. Set up your environment by exporting the IBM Cloud API Key and RHEL Subscription Password, as shown in Example 3-69.

*Example 3-69 Setting environment variables*

---

```
$ set +o history
$ export IBMCLOUD_API_KEY='<your API key>'
$ export RHEL_SUBS_PASSWORD='<your RHEL subscription password>'
$ set -o history
```

---

4. Run the create command:

```
$ ./openshift-install-powervs create
```

The script sets up the required tools and runs in interactive mode, which prompts for inputs.

5. When the command in step 4 on page 171 completes, it prints the cluster access information. Log in to bastion:

```
'ssh -i automation/data/id_rsa root@XXX.XXX.XXX.XXX'
```

To access the cluster on a local system when using 'oc', run the following command:

```
'export KUBECONFIG=/root/ocp-install-dir/automation/kubeconfig'
```

6. Access the Red Hat OpenShift web-console by navigating to the following URL:

```
https://console-openshift-console.apps.test-ocp6f2c.ibm.com
```

7. Log in to the console by running the following command:

```
user: "kubeadmin", and password: "MHvmI-z5nY8-CBFKF-hmCDJ"
```

8. To access your new cluster without defining a DNS for your Red Hat OpenShift Cluster, add these lines to your local system 'hosts' file:

```
XXX.XXX.XXX.XXX.,api.test-ocp6f2c.ibm.com  
console-openshift-console.apps.test-ocp6f2c.ibm.com  
integrated-oauth-server-openshift-authentication.apps.test-ocp6f2c.ibm.com  
oauth-openshift.apps.test-ocp6f2c.ibm.com  
prometheus-k8s-openshift-monitoring.apps.test-ocp6f2c.ibm.com  
grafana-openshift-monitoring.apps.test-ocp6f2c.ibm.com  
bolsilludo.apps.test-ocp6f2c.ibm.com
```

### **Advanced usage**

Before running the script, you can choose to override some environment variables, depending on your requirements. By default, Red Hat OpenShift 4.12 is installed. If you want to install 4.11, then export the following variables:

```
$ export RELEASE_VER="4.11"  
ARTIFACTS_VERSION: Tag/Branch (eg: release-4.11, v4.11, main) of ocp4-upi-powervs  
repository. The default is "main".  
$ export ARTIFACTS_VERSION="release-4.11"
```

### **Non-interactive mode**

You can avoid interactive mode by placing the required input files into the installation directory. The required input files are the following ones:

- ▶ SSH key files (file names `id_rsa` and `id_rsa.pub`)
- ▶ Terraform vars file (file name `var.tfvars`).

An example is shown in Example 3-70.

#### *Example 3-70 Example var.tfvars file*

---

```
ibmcloud_region = "syd"  
ibmcloud_zone = "syd04"  
service_instance_id = "123456abc-xzz-2223434343"  
rhel_image_name = "rhel-83-12062022"  
rhcos_image_name = "rhcos-412-02012023"  
network_name = "ocpnet"  
openshift_install_tarball =  
"https://mirror.openshift.com/pub/openshift-v4/ppc64le/clients/ocp/stable-4.12/ope  
nshift-install-linux.tar.gz"
```

```

openshift_client_tarball =
"https://mirror.openshift.com/pub/openshift-v4/ppc64le/clients/ocp/stable-4.12/ope
nshift-client-linux.tar.gz"
cluster_id_prepatch = "test-ocp"
cluster_domain = "xip.io"
storage_type = "nfs"
volume_size = "300"
bastion = {memory = "16", processors = "1", "count" = 1}
bootstrap = {memory = "32", processors = "0.5", "count" = 1}
master = {memory = "32", processors = "0.5", "count" = 3}
worker = {memory = "32", processors = "0.5", "count" = 2}
rhel_subscription_username = "mysubscription@email.com"
rhel_subscription_password = "mysubscriptionPassword"
...

```

You can also pass a custom Terraform variables file by using the option `-var-file <filename>` to the script. You can also use the option `-var "key=value"` to pass a single variable. If the same variable is used more than once, then precedence is from left (low) to right (high).

For a flowchart about how the script and process work, see Figure 3-23.

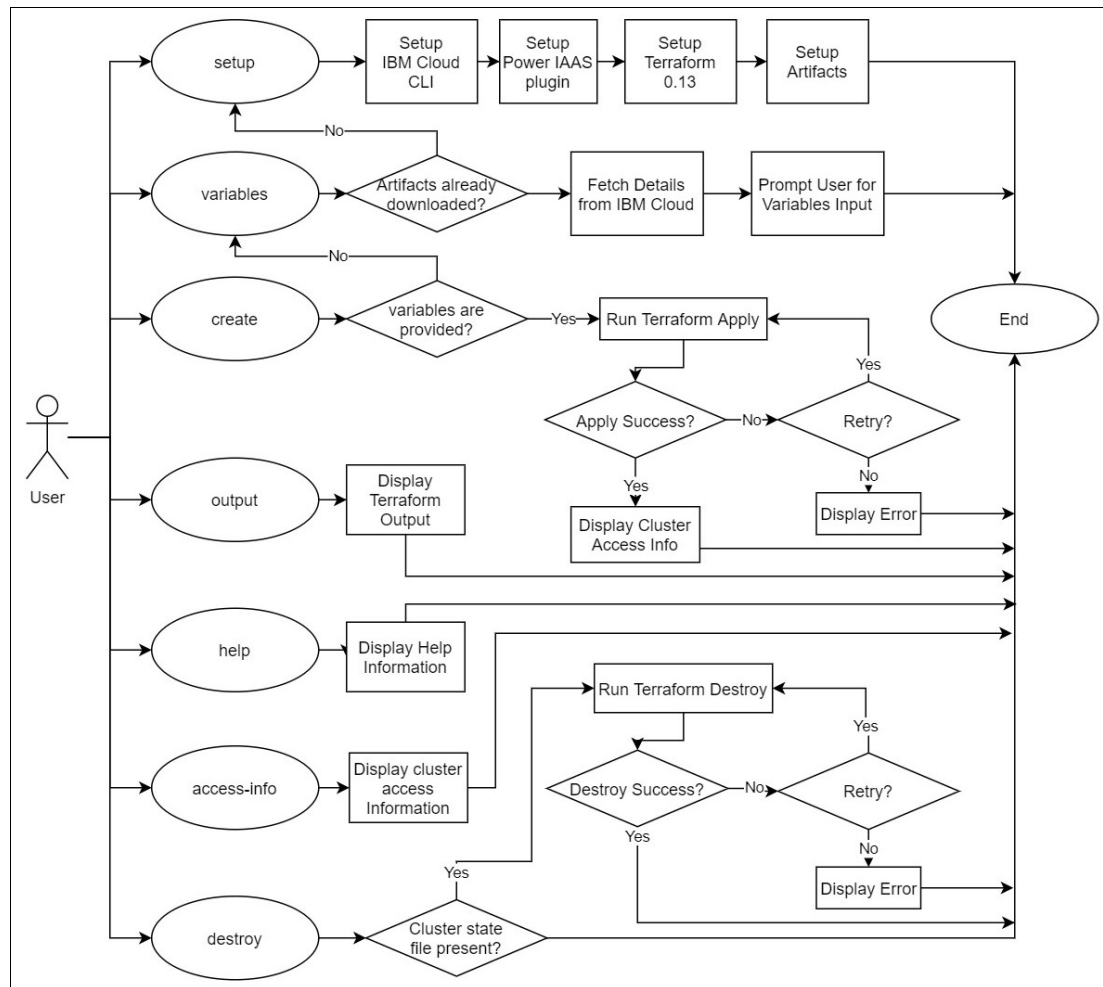


Figure 3-23 An openshift-install-powervs script interactions flowchart

## Deploying an Red Hat OpenShift Cluster by using PowerVC

This section provides a methodology for installing a Red Hat OpenShift cluster on your Power infrastructure that is managed by IBM PowerVC. The process is similar to the one that is used to install Red Hat OpenShift on Power Systems Virtual Server, as described in “Installing Red Hat OpenShift on Power Systems Virtual Server by using Ansible” on page 169 and also uses Terraform. Terraform 1.2.0 and later is required.

### *Installing Terraform and providers for a Power environment*

Before you install your Red Hat OpenShift cluster using this process, first install Terraform for your Power environment by completing the following steps:

1. Download and install the latest Terraform binary file for Linux/ppc64le from [GitHub](#).
2. Download the required Terraform providers for Power into your TF project directory, as shown in Example 3-71.

#### *Example 3-71 Downloading and installing Terraform*

---

```
$ cd <path_to_TF_project>
$ mkdir -p ./providers
$ curl -fsSL
https://github.com/ocppower-automation/terraform-providers-power/releases/download
/v0.11/archive.zip -o archive.zip
$ unzip -o ./archive.zip -d ./providers
$ rm -f ./archive.zip
Initialize Terraform at your TF project directory:
$ terraform init --plug-in-dir ./providers
```

---

The Ansible code for this process is provided by the [ocp4-upi-powervm project](#), which provides Terraform based automation code to help the deployment of Red Hat OpenShift Container Platform 4.x on PowerVM systems that are managed by PowerVC. This project uses the same Ansible playbook internally for Red Hat OpenShift Container Platform deployment on PowerVM LPARs that are managed through PowerVC.

If you are not using PowerVC but instead are using stand-alone PowerVM LPAR management, then [this guide](#) describes the process of using the Ansible playbook to set up a helper node (bastion) to simplify the Red Hat OpenShift Container Platform deployment.

### **PowerVC prerequisites**

As part of the installation process, create a Red Hat CoreOS image and an RHEL 8.2 (or later) image in PowerVC. The RHEL 8.x image is installed on the bastion node and the Red Hat CoreOS image is installed on the bootstrap, master, and worker nodes.

- ▶ For Red Hat CoreOS image creation, complete the steps that are documented in this [document](#).
- ▶ For RHEL image creation, complete the steps that are in this [document](#). You may either create a new image from ISO or use a similar method to the one that is used for the Red Hat CoreOS option.

### **Compute templates**

Create compute templates for the bastion, bootstrap, master, and worker nodes. Here are best practice LPAR configurations:

<b>Bootstrap</b>	Two vCPUs, 16 GB of RAM, and 120 GB of disk.
<b>Master</b>	Two vCPUs, 32 GB of RAM, and 120 GB of disk.
<b>Worker</b>	Two vCPUs, 32 GB of RAM, and 120 GB of disk.

<b>Bastion</b>	Two vCPUs, 16 GB of RAM, and 200 GB of disk. Increase the worker and bastion node settings based on application requirements.
<b>PowerVM</b>	LPARs by default use SMT8, so with two vCPUs, the number of logical CPUs as seen by the OS is 16 (two vCPUs x eight SMT).

### ***Downloading the automation code***

Use git to clone the deployment code when working off the master branch by using these commands:

```
$ git clone https://github.com/ocppower-automation/ocp4-upi-powervm.git
$ cd ocp4_upi_powervm
```

You can find instructions in the following file (if you are in the code directory):

```
ocp4-upi-powervm
```

### ***Setting the Terraform variables***

Update the var.tfvars file based on your environment. A description of the variables is available [at this link](#). You can use environment variables for sensitive data that should not be saved to disk, as shown in Example 3-72.

*Example 3-72 Example of setting environment variables*

---

```
$ set +o history
$ export POWERVC_USERNAME=xxxxxxxxxxxxxxxxx
$ export POWERVC_PASSWORD=xxxxxxxxxxxxxxxxx
$ export RHEL_SUBS_USERNAME=xxxxxxxxxxxxxxxxx
$ export RHEL_SUBS_PASSWORD=xxxxxxxxxxxxxxxxx
$ set -o history
```

---

### ***Starting an installation***

Run the following commands from within the directory:

```
$ terraform init
$ terraform apply -var-file var.tfvars
```

If you use environment variables for sensitive data, run the following commands instead:

```
$ terraform init
$ terraform apply -var-file var.tfvars -var user_name="$POWERVC_USERNAME" -var
password="$POWERVC_PASSWORD" -var rhel_subscription_username="$RHEL_SUBS_USERNAME"
-var rhel_subscription_password="$RHEL_SUBS_PASSWORD"
```

Wait for the installation to complete. It can take around 40 minutes to complete provisioning.

If the installation is successful, the cluster details are printed as shown in Example 3-73.

*Example 3-73 Output from the Terraform installation*

---

```
bastion_private_ip = 192.168.25.171
bastion_public_ip = 16.20.34.5
bastion_ssh_command = ssh -i data/id_rsa root@16.20.34.5
bootstrap_ip = 192.168.25.182
cluster_authentication_details = Cluster authentication details are available in
16.20.34.5 under ~/openstack-upi/auth
cluster_id = test-cluster-9a4f
etc_hosts_entries =
```

```
16.20.34.5 api.test-cluster-9a4f.mydomain.com
console-openshift-console.apps.test-cluster-9a4f.mydomain.com
integrated-oauth-server-openshift-authentication.apps.test-cluster-9a4f.mydomain.c
om oauth-openshift.apps.test-cluster-9a4f.mydomain.com
prometheus-k8s-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
grafana-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
bolsilludo.apps.test-cluster-9a4f.mydomain.com
```

```
install_status = COMPLETED
master_ips = [
    "192.168.25.147",
    "192.168.25.176",
]
oc_server_url = https://test-cluster-9a4f.mydomain.com:6443
storageclass_name = nfs-storage-provisioner
web_console_url =
https://console-openshift-console.apps.test-cluster-9a4f.mydomain.com
worker_ips = [
    "192.168.25.220",
    "192.168.25.134",
]
```

---

If you are using a wildcard domain name like `nip.io` or `xip.io`, then `etc_host_entries` is empty, as shown in Example 3-74.

*Example 3-74 Output when using a wildcard domain name*

---

```
bastion_private_ip = 192.168.25.171
bastion_public_ip = 16.20.34.5
bastion_ssh_command = ssh -i data/id_rsa root@16.20.34.5
bootstrap_ip = 192.168.25.182
cluster_authentication_details = Cluster authentication details are available in
16.20.34.5 under ~/openstack-upi/auth
cluster_id = test-cluster-9a4f
etc_hosts_entries =
install_status = COMPLETED
master_ips = [
    "192.168.25.147",
    "192.168.25.176",
]
oc_server_url = https://test-cluster-9a4f.16.20.34.5.nip.io:6443
storageclass_name = nfs-storage-provisioner
web_console_url =
https://console-openshift-console.apps.test-cluster-9a4f.16.20.34.5.nip.io
worker_ips = [
    "192.168.25.220",
    "192.168.25.134",
]
```

---

This information can be retrieved anytime by running the following command from the root folder of the code:

```
$ terraform output
```

If there was any errors, rerun `apply`. For more information about potential issues and workarounds, see [known issues](#).



## Postinstallation

After the deployment completes successfully, you can safely delete the bootstrap node. This step is optional but a best practice to free used resources. To delete the bootstrap node, change the count value to 0 in the `<bootstrap map>` variable and re-run the `apply` command.

## Creating an API and Ingress DNS records

If your `cluster_domain` is one of the online wildcard DNS domains (`nip.io`, `xip.io`, or `sslip.io`), skip this section. For all other domains, you can use one of the following options:

- ▶ Add entries to your DNS server by using the following general format:

```
api.<cluster_id>. IN A <bastion_public_ip>
*.apps.<cluster_id>. IN A <bastion_public_ip>
```

You need the `bastion_public_ip` and `cluster_id`, which were printed at the end of your successful installation, or you can retrieve those values anytime by running `terraform output` from the installation directory.

- ▶ Add entries to your client system hosts file.

**Tip:** For Linux and Mac OS, the hosts file is `/etc/hosts`; for Windows, it is `c:\Windows\System32\Drivers\etc\hosts`.

The general format is shown in Example 3-75. The entries for your installation are printed at the end of your successful installation. Alternatively, you can retrieve the entries anytime by running `terraform output` from the installation directory. Append these values to the host file.

### Example 3-75 Entries for the hosts file

---

```
<bastion_public_ip> api.<cluster_id>
<bastion_public_ip> console-openshift-console.apps.<cluster_id>
<bastion_public_ip>
integrated-oauth-server-openshift-authentication.apps.<cluster_id>
<bastion_public_ip> oauth-openshift.apps.<cluster_id>
<bastion_public_ip> prometheus-k8s-openshift-monitoring.apps.<cluster_id>
<bastion_public_ip> grafana-openshift-monitoring.apps.<cluster_id>
<bastion_public_ip> <app name>.apps.<cluster_id>
```

---

## Cluster access

After your cluster is running, you can log in to the cluster by using the Red Hat OpenShift login credentials in the bastion host. The location is printed at the end of a successful installation. You can retrieve the credentials anytime by running `terraform output` from the install directory. An example is shown in Example 3-76.

### Example 3-76 Login credentials for the Red Hat OpenShift cluster

---

```
bastion_public_ip = 16.20.34.5
bastion_ssh_command = ssh -i data/id_rsa root@16.20.34.5
cluster_authentication_details = Cluster authentication details are available in
16.20.34.5 under ~/openstack-upi/auth
```

---

There are two files under `~/openstack-upi/auth`:

- ▶ `kubeconfig`: Can be used for CLI access
- ▶ `kubeadmin-password`: The password for the `kubeadmin` user, which can be used for CLI and UI access

**Note:** Help ensure that you securely store the Red Hat OpenShift cluster access credentials. If you want, you can delete the access details from the bastion node after securely storing them elsewhere.

You can copy the access details to your local system or a secure password vault by running the following command:

```
$ scp -r -i data/id_rsa root@158.175.161.118:~/openstack-upi/auth/\*
```

### ***Cleaning up***

If you are finished with your cluster and want to destroy it, run the following command:

```
terraform destroy -var-file var.tfvars
```

This command helps ensure that all resources are cleaned up. Do not manually clean up your environment unless both of the following items are true:

- ▶ You know what you are doing.
- ▶ Something went wrong with an automated deletion.

## **Deploying an Red Hat OpenShift cluster on a Power based KVM environment**

The [ocp4-upi-kvm project](#) provides Terraform based automation code to help the deployment of Red Hat OpenShift Container Platform 4.x on KVM VMs by using `libvirt`. This project uses the Ansible playbook ([ocp4-helpernode](#)) to set up a helper node (bastion) for Red Hat OpenShift Container Platform deployment. The Ansible script creates a helper node for running the installation. For more information about the helper node, see [GitHub](#).

### ***Automation host prerequisites***

The automation must run from a system with internet access, which can be your laptop or a VM with public internet connectivity. This automation code has been tested on the following 64-bit OSs:

- ▶ Linux (preferred)
- ▶ Mac OSX (Darwin)

The automation host installation is documented at [GitHub ocp-power](#). The installation covers a Terraform installation and creating installation images for Red Hat CoreOS and RHEL.

### ***LibVirt prerequisites***

KVM virtualization relies on `libvirt` to work, so as a prerequisite for installing Red Hat OpenShift on KVM, complete the steps at this [GitHub repository](#).

### ***Downloading the automation code***

Use the following `git` command to clone the deployment code when working off the master branch:

```
git clone https://github.com/ocppower-automation/ocp4-upi-kvm.git
cd ocp4_upi_kvm
```

All further instructions assume that you are in the code directory `ocp4-upi-kvm`.

### ***Setting the Terraform variables***

Update the `var.tfvars` file based on your environment. A description of the variables is available at [How to use var.tfvars](#). You can use environment variables for sensitive data that should not be saved to disk, as shown in Example 3-77 on page 179.

*Example 3-77 Setting the environment variables for sensitive data*

---

```
$ export RHEL_SUBS_USERNAME=xxxxxxxxxxxxxxxxx
$ export RHEL_SUBS_PASSWORD=xxxxxxxxxxxxxxxxx
$ set -o history
$ set +o history
```

---

**Starting the installation**

Run the following commands from within the directory.

```
$ terraform init
$ terraform apply -var-file var.tfvars
```

If you use environment variables for sensitive data, then run the following commands instead:

```
$ terraform init
$ terraform apply -var-file var.tfvars -var
rhel_subscription_username="$RHEL_SUBS_USERNAME" -var
rhel_subscription_password="$RHEL_SUBS_PASSWORD"
```

Wait for the installation to complete. It can take around 40 minutes to complete provisioning.

If the installation is successful, the cluster details are printed, as shown in Example 3-78.

*Example 3-78 Cluster details*

---

```
bastion_ip = 192.168.61.2
bastion_ssh_command = ssh root@192.168.61.2
bootstrap_ip = 192.168.61.3
cluster_id = test-cluster-9a4f
etc_hosts_entries =
192.168.61.2 api.test-cluster-9a4f.mydomain.com
console-openshift-console.apps.test-cluster-9a4f.mydomain.com
integrated-oauth-server-openshift-authentication.apps.test-cluster-9a4f.mydomain.c
om oauth-openshift.apps.test-cluster-9a4f.mydomain.com
prometheus-k8s-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
grafana-openshift-monitoring.apps.test-cluster-9a4f.mydomain.com
bolsilludo.apps.test-cluster-9a4f.mydomain.com

install_status = COMPLETED
master_ips = [
  "192.168.61.4",
  "192.168.61.5",
  "192.168.61.6",
]
oc_server_url = https://api.test-cluster-9a4f.mydomain.com:6443/
storageclass_name = nfs-storage-provisioner
web_console_url =
https://console-openshift-console.apps.test-cluster-9a4f.mydomain.com
worker_ips = []
```

---

These details can be retrieved anytime by running the following command from the root folder of the code:

```
$ terraform output
```

## Postinstallation

After installation, complete the following tasks:

- ▶ Delete the bootstrap node. This step is optional but a best practice to free the resources that were used. The process is described in “Cleaning up” on page 178.
- ▶ Configure the DNS, as described in “Creating an API and Ingress DNS records” on page 177.

## 3.4.6 IBM Power Hardware Management Console as an Ansible managed client

HMCs can be Ansible clients, either by using the HMC Collection that connects to the HMC by using the HTTPS API, or by running the `ansible.builtin.shell` extension by using `cmd/ssh`. This section describes some of the setup considerations for setting up your HMC as an Ansible client.

### Defining a user for Ansible

It is not a best practice to use `hscroot` to run your Ansible scripts. This section provides a quick guide to creating a user for your Ansible script and provides the correct roles and access capabilities.

To set up your user and roles, complete the following steps:

1. Log in to your HMC with a user that has admin rights and click **Users and Security**.
2. Select **Users and Roles**.
3. Click **Manage User Profiles and Access**, as shown in Figure 3-24.

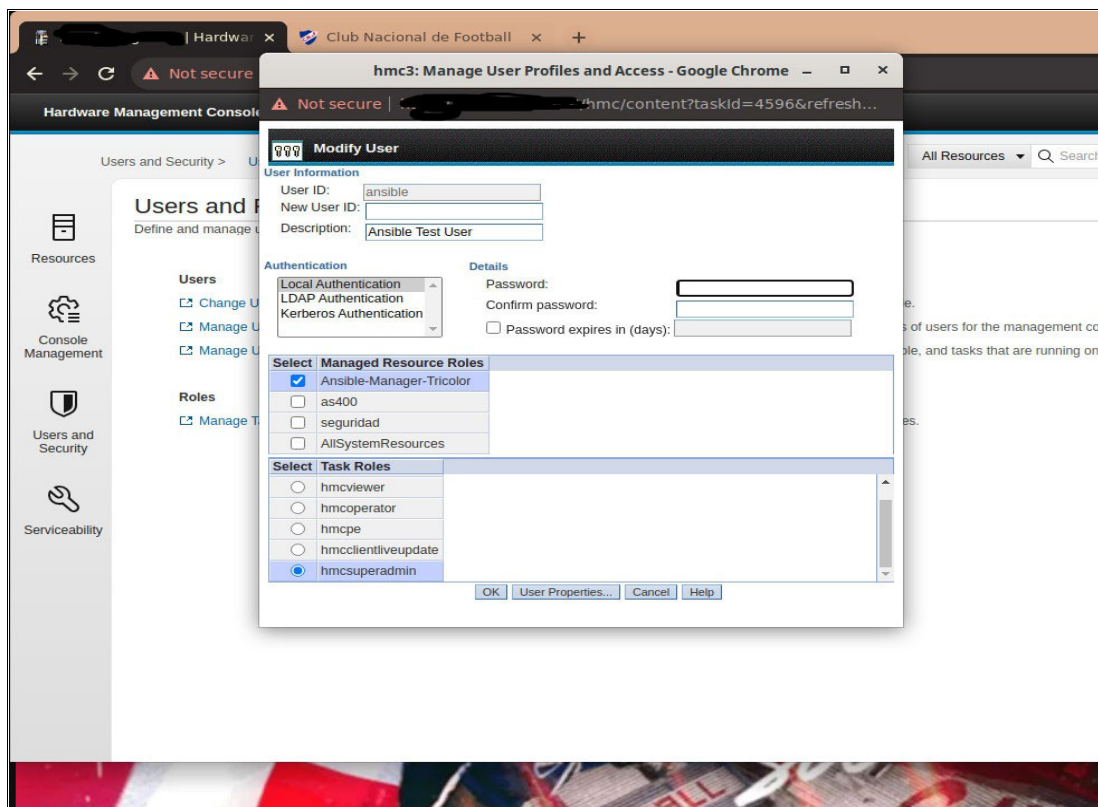


Figure 3-24 Creating a user role in the HMC

4. Complete the required fields.
5. Select **Managed Resource Roles**.
6. Select **Task Roles**. You can choose to create roles based on your specific requirements.
7. Click **User Properties** to enable remote access, as shown in Figure 3-25.

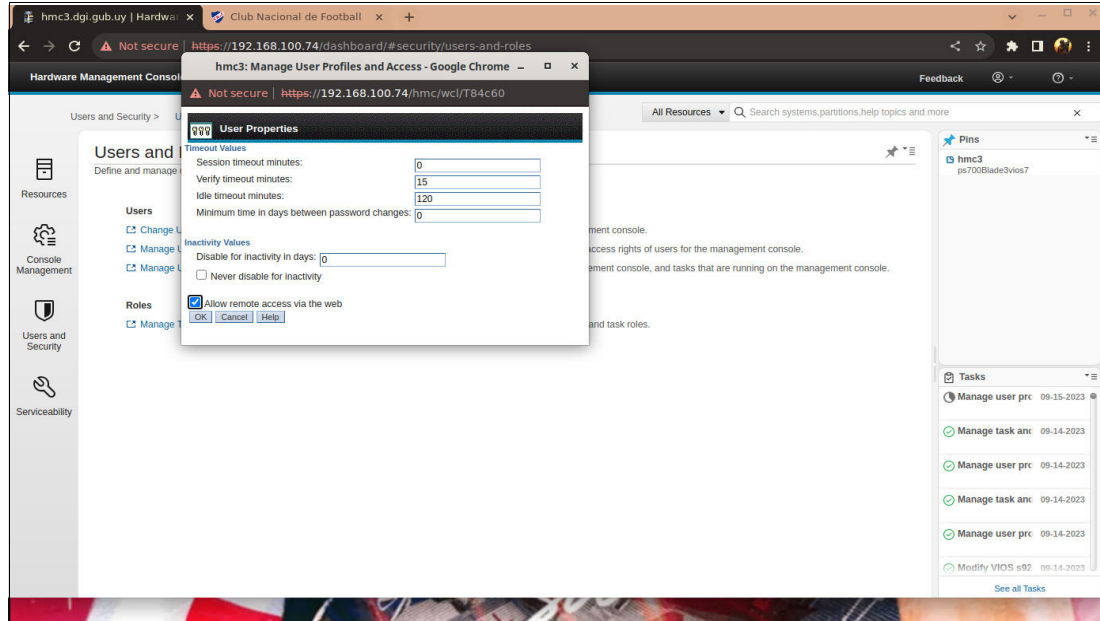


Figure 3-25 Enabling remote access for a user

8. Depending on your security requirements, you might need to modify the access privileges that define what resources (frames and LPARs) can be managed by your Ansible user. This information is defined in the user roles, and for each role you can define access to a reduced set of frames, and on those frames, a reduced set of machines. To illustrate this task, in Figure 3-26 we created a role that is allowed to manage only a reduced set of frames and LPARs.

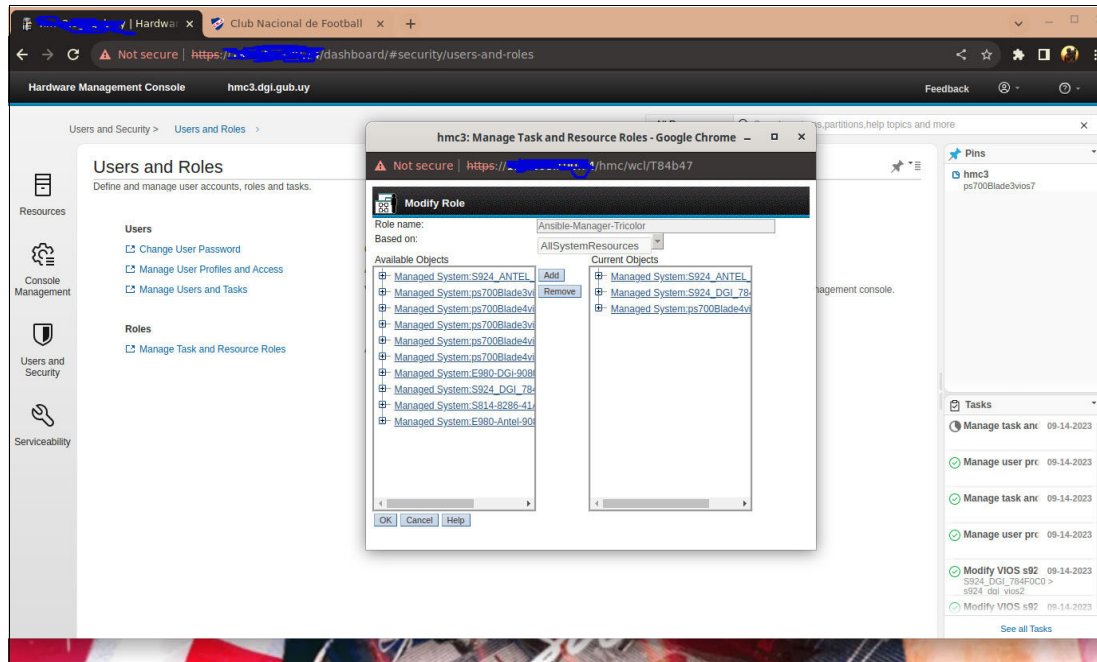


Figure 3-26 Creating a reduced role

9. If you define a role after creating the user, you can modify the user and select the new role, as shown in Figure 3-27.

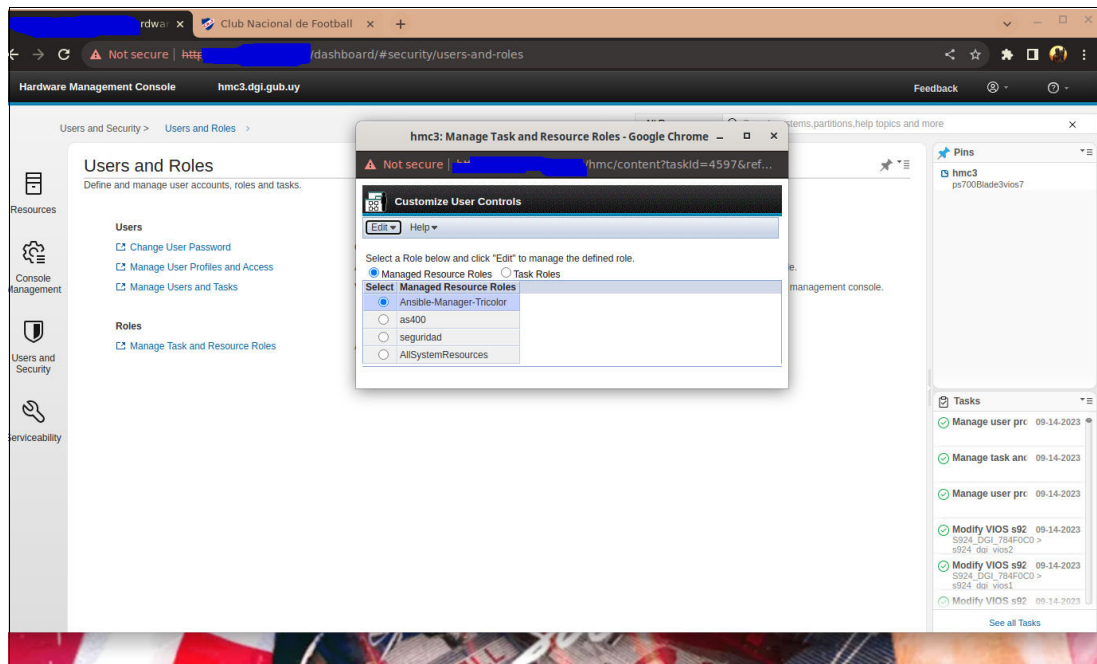


Figure 3-27 Selecting a role for a user

You set up a user with a set of permissions to run your Ansible scripts through SSH or through the `hmc ibm power` collection.

## Installing the collection

To install the collection on your Ansible Controller, complete the installation instructions at this [GitHub repository](#).

Along with the installation guide, here are some hints and tips that we collected during our testing:

1. Before you run the installation, make sure that you set up `python3.9` as the default for `python` and `pip3`, as shown in Example 3-79.

### *Example 3-79 The Python installation directory*

---

```
[root@ansible-AAP-redbook ~]# alternatives --list | grep -i python
python          auto    /usr/libexec/no-python
python3         manual /usr/bin/python3.9
root@ansible-AAP-redbook ~]# ls -lrt /usr/bin/python3
lrwxrwxrwx. 1 root root 25 Nov  3 2021 /usr/bin/python3 ->
/etc/alternatives/python3
[root@ansible-AAP-redbook ~]# ls -lrt /usr/bin/pip3
lrwxrwxrwx. 1 root root 22 Nov  3 2021 /usr/bin/pip3 -> /etc/alternatives/pip3
[root@ansible-AAP-redbook ~]#
```

---

2. Run `ansible-galaxy`.

If you are behind a proxy, you might need to run the `ansible` command with the `--ignore-certs` option. The collection installation progress is output to the console. Note the location of the installation so that you can review other content that is included with the collection, such as the sample playbook. The collection installation process is shown in Example 3-80.

### *Example 3-80 Installing a collection on the Ansible Controller node*

---

```
[root@ansible-AAP-redbook ~]# ansible-galaxy collection install ibm.power_hmc
--ignore-certs
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ibm-power_hmc-1.8.0.tar.gz to
/root/.ansible/tmp/ansible-local-8811f_xjfbx/tmpgzq9rzjl/ibm-power_hmc-1.8.0-hi_q
msj1
Installing 'ibm.power_hmc:1.8.0' to
'/root/.ansible/collections/ansible_collections/ibm/power_hmc'
ibm.power_hmc:1.8.0 was installed successfully
[root@ansible-AAP-redbook ~]#
```

---

If you want to install on a special directory, run `install -p` to specify the installation directory:

```
[root@ansible-AAP-redbook ~]# ansible-galaxy collection install ibm.power_hmc
--ignore-certs -p /home/ansible/collections
```

If the installation completes successfully, you can check the installation directory. (Instead of /root, the path includes the user directory that is tied to the user that you used to run install).

```
[root@ansible-AAP-redbook ~]# pwd
/root/.ansible/collections/ansible_collections/ibm/power_hmc/
```

Example 3-81 shows the content of the installation directory in our example installation.

*Example 3-81 Structure of the installation directory*

---

```
[root@ansible-AAP-redbook power_hmc]# ls -lrt
total 88
-rw-r--r--. 1 root root  947 Aug 22 15:14 MANIFEST.json
-rw-r--r--. 1 root root 35149 Aug 22 15:14 LICENSE
-rw-r--r--. 1 root root 16874 Aug 22 15:14 FILES.json
-rw-r--r--. 1 root root  2824 Aug 22 15:14 CONTRIBUTING.md
drwxr-xr-x. 5 root root   77 Aug 22 15:14 plug-ins
-rw-r--r--. 1 root root  263 Aug 22 15:14 MAINTAINERS.md
drwxr-xr-x. 2 root root   30 Aug 22 15:14 collections
-rw-r--r--. 1 root root  3227 Aug 22 15:14 CODE_OF_CONDUCT.md
drwxr-xr-x. 4 root root   32 Aug 22 15:14 tests
drwxr-xr-x. 4 root root   53 Aug 22 15:14 docs
-rw-r--r--. 1 root root    5 Aug 22 15:14 requirements.txt
drwxr-xr-x. 2 root root   25 Aug 22 15:14 meta
-rw-r--r--. 1 root root  1352 Aug 22 15:14 README.md
drwxr-xr-x. 3 root root   59 Aug 22 15:44 context
-rw-r--r--. 1 root root  144 Aug 22 16:51 execution-environment.yml
drwxr-xr-x. 2 root root  4096 Sep 15 11:55 playbooks
[root@ansible-AAP-redbook power_hmc]#
```

---

## Connecting to the HMC

The collection connects to the HMC by using the HMC API, but you can also use the Ansible native SSH connection to run HMC commands.

Example 3-82 shows an example of using the Ansible built-in shell to run CLI commands on the HMC.

*Example 3-82 Using the Ansible built-in shell to manage HMCs*

---

```
[root@ansible-AAP-redbook playbooks]# cat inventory
[hmcs]
hmc3
hmc4
hmc5

[hmcs:vars]
ansible_user=ansible
hmc_password=xxxxxx

[root@ansible-AAP-redbook playbooks]# cat hmc_lpar_limited.yml
---
- name: HMC List partition
  hosts: hmc3
  collections:
    - ibm.power_hmc
  connection: local
```



```

vars:
  curr_hmc_auth:
    username: "{{ ansible_user }}"
    password: "{{ hmc_password }}"

tasks:
  - name: Get information about the Frames Installed on the HMC
    ansible.builtin.shell:
      cmd: "sshpass -p {{ hmc_password }} ssh {{ ansible_user }}@{{
inventory_hostname }} lssyscfg -r sys"
      register: config

  - name: Filter Output to get just the Frame Name
    set_fact:
      filtered_output: "{{ config.stdout | regex_findall('^name=(.*?),', '\\1')
| join(',') }}"

  - name: Show Filtered Output
    debug:
      var: filtered_output

  - name: Save Filtered Output to a File for Later use
    ansible.builtin.copy:
      content: "{{ filtered_output }}"
      dest: "/var/ansible/output/newout_{{ inventory_hostname }}.txt"

  - name: List all the LPARs in the selected Frames
    ansible.builtin.shell:
      cmd: " sshpass -p {{ hmc_password }} ssh {{ ansible_user }}@{{
inventory_hostname }} lssyscfg -r lpar -m {{ item }}"
      with_items: "{{ filtered_output.split(',') }}"
      register: lpar_info

  - name: Save the Output, LPAR List for each Frame
    ansible.builtin.copy:
      content: "{{ lpar_info }}"
      dest: "/var/ansible/output/Lparout.txt"

  - name: Show LPAR Information
    debug:
      var: item.stdout_lines
      with_items: "{{ lpar_info.results }}"

```

```
[root@ansible-AAP-redbook playbooks]#
```

---

In our example playbook execution, the inventory has three HMCs, but we created only the user with special permits in one of them (hmc3). When we ran the playbook, the others failed because the user does not exist. The output from the execution is shown in Example 3-83.

*Example 3-83 Playbook execution*

---

```
[root@ansible-AAP-redbook playbooks]# ansible-playbook -i inventory
hmc_lpar_limited.yml
```

```
PLAY [HMC List partition] *****
```

```
TASK [Gathering Facts] *****
ok: [hmc3]
ok: [hmc5]
ok: [hmc4]
```

```
TASK [Get Information about the Frames Installed on the HMC] *****
changed: [hmc3]
fatal: [hmc5]: FAILED! => {"changed": true, "cmd": "sshpass -p XXXXXX ssh
ansible@hmc5 lssyscfg -r sys", "delta": "0:00:01.867500", "end": "2023-09-18
15:42:07.700107", "msg": "non-zero return code", "rc": 5, "start": "2023-09-18
15:42:05.832607", "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines":
[]}
fatal: [hmc4]: FAILED! => {"changed": true, "cmd": "sshpass -p XXXXXX ssh
ansible@hmc4 lssyscfg -r sys", "delta": "0:00:01.843471", "end": "2023-09-18
15:42:07.787177", "msg": "non-zero return code", "rc": 5, "start": "2023-09-18
15:42:05.943706", "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines":
[]}
```

```
TASK [Filter Output to get just the Frame Name] *****
ok: [hmc3]
```

```
TASK [Show filtered Output] *****
ok: [hmc3] => {
  "filtered_output": "S924_ANTEL_XXXX,S924_DGI_XXXX,ps700Blade4XXXX"
}
```

```
TASK [Save Filtered Output to a File for Later use] *****
ok: [hmc3]
```

```
TASK [List all the LPARs in the selected Frames] *****
changed: [hmc3] => (item=S924_ANTEL_XXXX)
changed: [hmc3] => (item=S924_DGI_784FOC0)
changed: [hmc3] => (item=ps700Blade4XXXX)
```

```
TASK [Save the Output, LPAR List for each Frame] *****
changed: [hmc3]
```

```
TASK [List with each LPAR Detailed Information] *****
ok: [hmc3] => {
  "lpar_info.stdout": "VARIABLE IS NOT DEFINED!"
}
```

```
TASK [Mostrar información de LPARs] *****
ok: [hmc3] => (item={'changed': True, 'stdout':
'name=ansibleRH8_1,lpar_id=27,lpar_env=aixlinux,state=Running,resource_config=1,os
_version=Unknown,logical_serial_num=XXXXX,default_profile=default_profile,curr_pro
file=default_profile,work_group_id=None,shared_proc_pool_util_auth=0,allow_perf_co
llection=0,power_ctrl_lpar_ids=None,boot_mode=norm,lpar_keylock=norm,auto_start=0,
redundant_err_path_reporting=0,rmc_state=inactive,rmc_ipaddr=,time_ref=0,lpar_avai
l_priority=127,desired_lpar_proc_compat_mode=default,curr_lpar_proc_compat_mode=PO
WER9_base,simplified_remote_restart_capable=0,sync_curr_profile=1,affinity_group_i
d=None,vtpm_enabled=0,migr_storage_vios_data_status=Data
Collected,migr_storage_vios_data_timestamp=Wed Sep 06 08:20:38 UTC
2023,powervm_mgmt_capable=0,pend_secure_boot=0,curr_secure_boot=0,keystore_kbytes=
0,virtual_serial_num=None\nname=ansibleRH8_2,lpar_id=28,lpar_env=aixlinux,state=Ru
```

```

nning,resource_config=1,os_version=Unknown,logical_serial_num=XXXXX,default_profile=
default_profile,curr_profile=default_profile,work_group_id=none,sha
...
...
...
start=0,redundant_err_path_reporting=0,rmc_state=active,rmc_ipaddr=XXXXXX,lpar_ava
il_priority=191,desired_lpar_proc_compat_mode=default,curr_lpar_proc_compat_mode=P
OWER7,sync_curr_profile=0,affinity_group_id=none,migr_storage_vios_data_status=una
vailable,migr_storage_vios_data_timestamp=unavailable"
    ]
}
PLAY RECAP *****
hmc3          : ok=9    changed=3    unreachable=0    failed=0
skipped=0     rescued=0    ignored=0
hmc4          : ok=1    changed=0    unreachable=0    failed=1
skipped=0     rescued=0    ignored=0
hmc5          : ok=1    changed=0    unreachable=0    failed=1
skipped=0     rescued=0    ignored=0

[root@ansible-AAP-redbook playbooks]#

```

This playbook is an example of using the SSH connection to run CLI commands. There are many other options that use the HMC collection that provide an extended set of features.

### Using the dynamic inventory plug-in

The dynamic inventory plug-in, `powervm_inventory`, helps to collect the inventory of all available VMs in the public or private cloud infrastructure that is managed by the HMC. This approach saves the administrator from maintaining a multitude of static inventory listings. This dynamic plug-in facilitates the consolidation of the partitions into various groups, and it can be further fine-tuned according to its property. You can now dynamically group and manage these partitions according to data center policy.

An example use case can be to apply specific patches only to a targeted version of AIX partition by using the respective group that is created by this dynamic inventory management plug-in. The playbook that is shown in Example 3-84 is an example of dynamically creating a group definition from the LPARs being managed. The playbook dynamically creates a group of all the running AIX partitions AIX 7.2,. This group can be used as an inventory input to perform patching or for OS upgrades.

*Example 3-84 Dynamic inventory example*

```

plug-in: ibm.power_hmc.powervm_inventory
hmc_hosts:
  - hmc: <hmc_host_name_or_IP>
    user: <hmc_username>
    password: <hmc_password>
filters:
  PartitionState: 'running'
groups:
  AIX_72: "'7.2' in OperatingSystemVersion"

```

## Patch management of HMC

The patch management module, `hmc_update_upgrade`, is designed to support all the update and upgrade requirements of HMC. The module supports centralized patch management. The upgrade files, service packs, and security fixes can be stored in SFTP, FTP, or NFS servers, or the HMC images can be kept on the Ansible control node and be configured as the image source for patch management. Example 3-85 shows a playbook for patching or upgrading the HMC.

### Example 3-85 Patching or upgrading the HMC

---

```
- name: Upgrade the HMC from 910 to 950
  hosts: hmcs
  collections:
    - ibm.power_hmc
  connection: local
  vars:
    curr_hmc_auth:
      username: <hmc_username>
      password: <hmc_password>

tasks:
  - name: Update the 910 HMC with 9.1.910.6 PTF
    hmc_update_upgrade:
      hmc_host: '{{ inventory_hostname }}'
      hmc_auth: '{{ curr_hmc_auth }}'
      build_config:
        location_type: ftp
        hostname: <FTP_Server_IP>
        userid: <FTP_Server_uname>
        passwd: <FTP_Server_pwd>
        build_file:
HMC9.1.910.6/2010170040/x86_64/MH01857-9.1.910.6-2010170040-x86_64.iso
      state: updated

  - name: Upgrade the 910 HMC with 950 image
    hmc_update_upgrade:
      hmc_host: '{{ inventory_hostname }}'
      hmc_auth: '{{ curr_hmc_auth }}'
      build_config:
        location_type: nfs
        hostname: <NFS_Hostname/IP>
        mount_location: <upgrade_img_mount_loc>
        build_file: /HMC9.2.950.0/2010230054/x86_64/network_install
      state: upgraded
```

---

In this snippet of a playbook, there are two tasks that are defined by using the `hmc_update_upgrade` module:

- ▶ The first task uses a Program Temporary Patch (PTF) image that is stored in an FTP server to upgrade the HMC running the 9.1.910 level.
- ▶ The second task upgrades the HMC to the 9.2.950 level by using an image that is stored in an NFS server.

To update an HMC, the task state should be defined as *updated*, and to upgrade an HMC, the task state should be defined as *upgraded*. Both states support NFS, FTP, SFTP, and disk (keeping the image in controller node) HMC image repositories.

Additional examples can be found [on this blog](#). More examples and samples are included in the collection and can be found in this [GitHub repository](#).





# Automated application deployment on IBM Power servers

One of the areas that can benefit the most from automation with Ansible is the area of application deployment. This notion is true for the applications that your application development team are building specifically for your business operations and for implementing common application environments like Oracle and SAP.

For application development, automation can help you build a modern continuous integration and continuous development (CI/CD) pipeline that provides a management structure around your development cycle and helps you be more agile by integrating application changes quickly and safely.

For application and database environments like Oracle and SAP, there are often multiple instances of those environments that must be deployed and maintained, often doing repetitive tasks like building virtual machines (VMs) and applying updates to your existing environments. Automation that uses Ansible can help your IT specialists perform those repetitive functions in an efficient, managed, and repeatable way so that you can do other tasks that provide better value to your business.

The following topics are described in this chapter:

- ▶ Deploying and managing applications by using Ansible on Power servers
- ▶ Automated application deployment on Power servers
- ▶ Deploying a simple Node.js application
- ▶ Orchestrating multitier application deployments
- ▶ Continuous integration and continuous deployment pipelines with Ansible
- ▶ Oracle DB automation on Power
- ▶ SAP automation

## 4.1 Deploying and managing applications by using Ansible on Power servers

Ansible can be used to automate many of the facets of your application environment, both on IBM Power and on other platforms. Ansible can help you automate your application development environments so that you can create pipelines that help manage the development of code, the testing of the code, and the integration of the new code into your application environment in a safe and efficient way.

Ansible can also automate the installation and management of many middleware and application products, such as Oracle Database, SAP NetWeaver and SAP HANA workloads, IBM Db2 databases, and Red Hat OpenShift deployments.

## 4.2 Automated application deployment on Power servers

The IBM Power platform is rapidly evolving, with major advances made across IBM AIX, IBM i, and Linux on Power. Hybrid multicloud demands consistency and agility from all these platforms. Today's IT administrators, developers, and quality assurance (QA) professionals want to streamline anything they can to save time and increase reliability.

### 4.2.1 Ansible content for IBM Power

IBM Power is a family of enterprise servers that helps transform your organization by delivering industry-leading resilience, scalability, and accelerated performance for the most sensitive, mission-critical workloads and next-generation artificial intelligence (AI) and edge solutions. The Power platform also uses open-source technologies that you use to run these workloads in a hybrid cloud environment with consistent tools, processes, and skills.

### 4.2.2 IBM AIX, IBM i, and Linux on Power collections for Ansible

The IBM Power AIX collection provides modules that can be used to manage configurations and deployments of Power AIX systems. Similar functions are provided by the Power IBM i collection and the Linux on Power collection. The content in these collections helps to manage applications and workloads running on IBM Power platforms as part of an enterprise automation strategy within the Ansible ecosystem. These collections are available from Ansible Galaxy with community support or through Red Hat Ansible Automation Platform with full support from Red Hat and IBM.

To show the power and utility of using Ansible to manage your applications in an IBM Power environment, we provide three use cases:

- ▶ Section 4.3, “Deploying a simple Node.js application” on page 193 shows how you can automate application deployment in an AIX environment.
- ▶ Section 4.4, “Orchestrating multitier application deployments” on page 194 points you to a tutorial on orchestrating a two-tier application with an application tier and a database tier.
- ▶ Section 4.5, “Continuous integration and continuous deployment pipelines with Ansible” on page 194 shows how to you can use Ansible to automate a CI/CD environment for an application in an IBM i environment.



## 4.3 Deploying a simple Node.js application

This section shows how to install a sample Node.js application on AIX and then start that application on the host that uses Ansible. The Ansible playbook to accomplish this task is shown in Example 4-1.

*Example 4-1 Installing the Node.js application and starting the application*

---

```
---
- hosts: all
  gather_facts: false
  collections:
    - ibm.power_ibmi

  vars:
    checkout_dir: 'tmp_nodejs'

  tasks:
    - name: Install Node js
      command: /Q0pensys/pkgs/bin/yum install nodejs10 -y
      ignore_errors: true

    - name: verify git has been installed
      stat:
        path: /Q0pensys/pkgs/bin/git
      register: git_stat

    - name: Install git if it is not there
      command: /Q0pensys/pkgs/bin/yum install git -y
      when: not git_stat.stat.exists

    - name: upgrade yum in case EC_POINT_copy error
      command: /Q0pensys/pkgs/bin/yum upgrade -y

    - name: create symlink for git command to use git module
      command: ln -fs /Q0pensys/pkgs/bin/git /usr/bin/git
      ignore_errors: true

    - name: set http.sslVerify for git
      command: 'git config --global http.sslVerify false'
      ignore_errors: true

    - name: clone repo
      git:
        repo: 'https://github.com/IBM/ibmi-oss-examples.git'
        dest: '{{ checkout_dir }}'

    - name: npm i
      shell: "(/Q0pensys/pkgs/lib/nodejs10/bin/npm i --scripts-prepend-node-path)"
      args:
        warn: false
        chdir: '{{ checkout_dir }}/nodejs/mynodeapp'
        executable: /usr/bin/sh

    - name: Start the demo application
```

```
    shell: "(nohup /Q0pensys/pkgs/lib/nodejs10/bin/node index.js >/dev/null 2>&1
&)"
    args:
      warn: false
      chdir: '{{ checkout_dir }}/nodejs/mynodeapp'
      executable: /usr/bin/sh
      async: 10
```

---

The playbook validates the required infrastructure components (git and yum) to retrieve the application and install it. Then, the playbook retrieves the application from the Git repository and has Ansible start the application on the server.

## 4.4 Orchestrating multitier application deployments

In the era of microservices, containers, Kubernetes, and DevOps, deployment of complex multitier systems in the public cloud is achieved with CI/CD pipelines. However, things become complicated when attempting to deliver applications onto private cloud or on-premises systems, where CI/CD is not owned by development or does not exist.

### 4.4.1 Orchestration in the world of Kubernetes

Deploying complex application systems is not a new problem. Over the past few decades, the need for automated configuration and management (orchestration) of software has been identified many times. In the operating systems (OSs) space, configuration management tools like Chef, Puppet, Salt, and Ansible orchestrate the configuration of OS-native applications.

From an application orchestration standpoint, many prominent players have emerged in the Kubernetes space, such as Helm, Operator Framework, Kustomize, Automation, Broker, and the Ansible Kubernetes module.

IBM developed a [step by step tutorial](#) that shows the deployment and orchestration of a sample WordPress application along with its dependent MySQL database that uses Ansible.

## 4.5 Continuous integration and continuous deployment pipelines with Ansible

Ansible can help you automate the CI/CD cycle for your application. The continuous integration (CI) part of the CI/CD involves automatic version control for your application code. As changes are made in your code, Ansible can automatically save those changes in a repository so that you know what changes were made and when. To enable the continuous deployment (CD) component, you must also have automation in your CI components that provides testing and validation of new versions of your code. You initially apply the new code in a test or QA environment before you move the code to production. Ansible can help with all of those processes.

## 4.5.1 CI/CD when using Ansible for IBM i

CI/CD stands as a pivotal pillar within the software development landscape. Ansible, an automation tool, harmonizes with these practices, aligning software development with a unified approach. CI revolves around frequent code integration, while CD automates deployment, helping ensure swift and reliable delivery. Ansible effectively supports these practices, orchestrating tasks and interactions through various stages.

For IBM i users, Ansible emerges as a versatile solution for crafting CI/CD pipelines. It interfaces with version control systems (VCSs) such as GitHub, facilitating tasks ranging from code validation to final deployment. Ansible utility spans beyond continuous testing, merging into the broader domain of development processes. This adaptability benefits both native applications and open-source software deployment on the IBM i platform.

The foundation of a CI/CD pipeline through Ansible takes shape as a series of interconnected stages. Starting with the development environment, Ansible playbooks automate tasks such as provisioning IBM i VMs, code retrieval, build and deployment, and branching for new development cycles. This initial phase lays the groundwork for efficient development practices.

In the second stage, developers play a hands-on role in code modifications and unit testing. In this phase, developers make code changes, conduct unit tests on the development IBM i system, commit code alterations to specific branches, and create pull requests. This phase demonstrates the collaborative essence of the development process.

As the CI/CD pipeline progresses, Ansible again takes center stage during the testing phase. Ansible playbooks orchestrate the setup of new IBM i VMs for testing, the installation of dependencies, cloning development branches, and the execution of comprehensive tests. Furthermore, Ansible helps with pull request approval and automated merge processes, helping ensure code integration. On test completion, Ansible manages environment cleanup for ongoing efficiency.

The final phase, deployment, encapsulates the core objective of delivering refined software to production environments. Ansible playbooks manage cloning the latest code from master branches to build systems, direct the build process, facilitate deployment to production systems, and maintain environment hygiene through cleanup procedures. This phase marks the realization of the CI/CD pipeline, where developed and tested code becomes accessible to users.

Figure 4-1 shows a diagram describing the four stages for CI/CD by using Ansible.

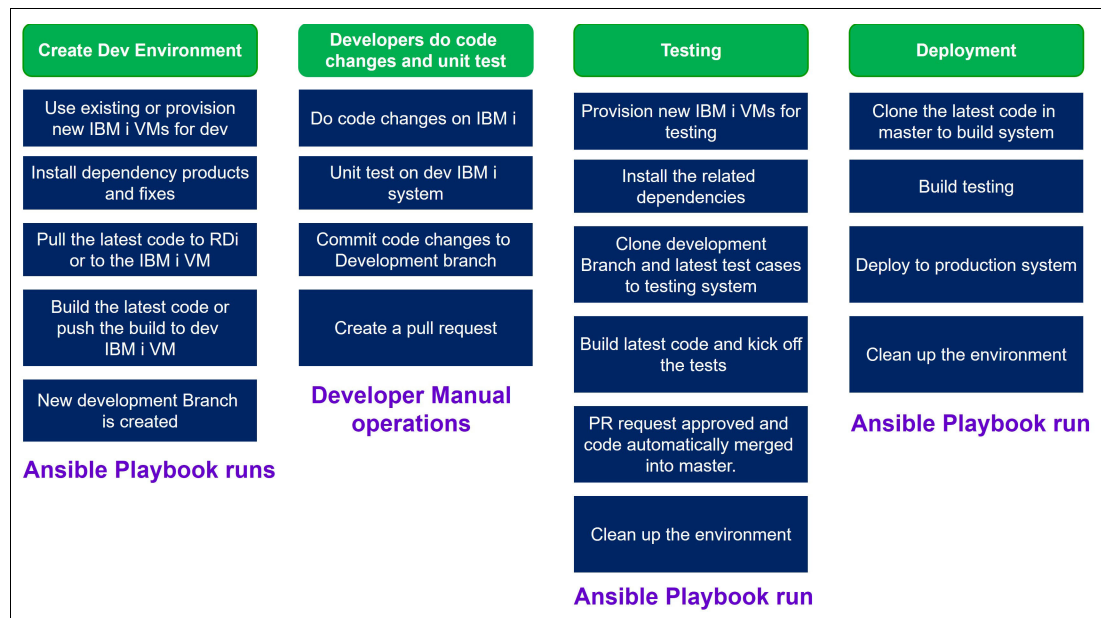


Figure 4-1 Integrated CI/CD Pipeline that is orchestrated by Ansible

To see some sample code showing how to automate your CI/CD processes with Ansible, see “Full cycle of the CI/CD process on IBM i with Ansible” on page 362.

**Note:** The role of Ansible within the CI/CD pipeline streamlines IBM i application development. Its automation capabilities extend across development environment creation, manual developer tasks, testing orchestration, and deployment management. By aligning with contemporary CI/CD practices, Ansible contributes to advancing IBM i software development with heightened speed, dependability, and agility.

## 4.6 Oracle DB automation on Power

There are three collections that are available for managing your Oracle Database with Ansible on AIX.

The single-node collection supports both Journaled File System (JFS) based installation and Automatic Storage Management (ASM)-based installation. The Oracle Real Application Cluster (RAC) collection supports a multi-layered installation process. Such layers include infrastructure provisioning with PowerVC, grid setup configuration and installation, and setup and installation of the database binary files.

If you are not using PowerVC to help you manage your environment, you can still use the collection to install and set up the grid and database installation. To do so, you must manually set up the nodes per Oracle RAC requirements.

The Oracle database administrator (DBA) collection provides a set of tools to enable DBAs to automate a wide range of their administrative activities, such as patching the database servers, creating database instances, managing users, jobs, and table spaces, and other operations.

For more information about the usage of each of these collections, see the following sections:

- ▶ 4.6.2, “Automating the deployment of a single-node Oracle database with Ansible” on page 197.
- ▶ 4.6.3, “Automating the deployment of Oracle RAC with Ansible” on page 201.
- ▶ 4.6.4, “Automating Oracle DBA operations” on page 214

### 4.6.1 Why businesses opt for AIX to host their databases

Section 1.4.8, “Key benefits of IBM Power compared to x86 servers” on page 29 describes why IBM Power is prominent in hosting applications and middleware in general. IBM Power offers better economics, and is highly flexible so that clients can build out a platform on which they can consolidate any number of applications and environments, including databases.

Working together to provide services for their common clients, IBM and Oracle established an alliance that shows a shared commitment to the success of those clients. As a result of this alliance, IBM and Oracle have over 80,000 joint clients that are provided with enhanced hardware and software support. This solution is enabled through an in-depth certification of Oracle database on AIX as a collective effort. The alliance provides a services practice, with a diamond partnership, as a result of extensive technology collaboration and cooperative client support.

For more information about IBM Power and AIX platform hosting an Oracle database, see *Oracle on IBM Power Systems*, SG24-8485.

### 4.6.2 Automating the deployment of a single-node Oracle database with Ansible

You can use Ansible to automate the deployment of a single-node Oracle database instance. It can be a JFS-based installation or an ASM-based installation. The hosting AIX logical partition (LPAR) and database storage volumes may be created beforehand as setup for this installation or can be deployed by Ansible, as described in Chapter 5, “Infrastructure as Code by using Ansible” on page 239.

Figure 4-2 shows a high-level overview of Oracle DB single-node deployment.

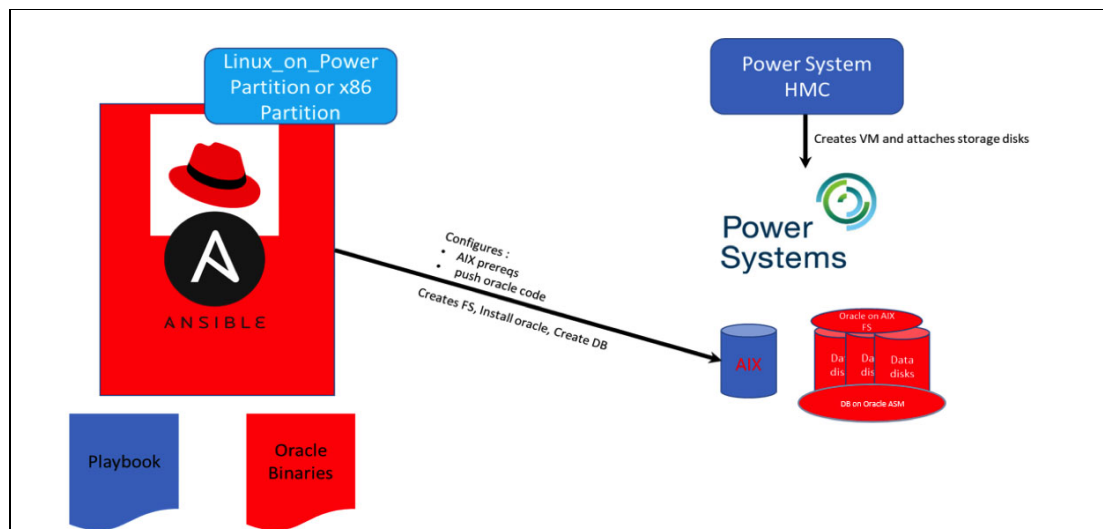


Figure 4-2 Single-node Oracle DB deployment topology<sup>1</sup>

The Ansible [power\\_aix\\_oracle](#) galaxy collection contains artifacts (roles, vars, configuration files, and playbooks) that are usable for the single-node Oracle database installation. Those artifacts have several prerequisites that must be met before using the collection.

## Single-node Oracle database installation prerequisites

The following prerequisites must be met before you run an Ansible playbook to automate the installation of a single-node Oracle database server:

1. A new LPAR running AIX 7.2 or 7.3.
  - a. The LPAR may be created by using Ansible Infrastructure as Code (IaC) methodology that is described in Chapter 5, “Infrastructure as Code by using Ansible” on page 239.
  - b. The rootvg disk should be at least 30 GB, out of which the /tmp file system should be 8 GB because it will be used for the Ansible remote location. Also, the paging space device must be adjusted to use rootvg.
  - c. Two disks of the following sizes:
    - i. 40 GB for a file system-based installation or 75 GB for an ASM-based installation.  
This disk is intended to host the database binary files for a file system-based installation types or the grid binary files for an ASM-based installation.
    - ii. At least 20 GB for the creation of a test Oracle database instance postinstallation.
  - d. The disks in step c should be clean from any old header data.  
To check the header information, use the following command:  

```
lquerypv -h /dev/hdiskX
```

  
If you must clear the disk’s PVID, use the following command:  

```
chdev -l hdiskX -a pv=clear
```

  
To clear the header data, use the following command:  

```
dd if=/dev/zero of=/dev/hdiskX bs=1024k count=100
```
2. At the time of writing, Oracle database 19c is supported on AIX. All `power_aix_oracle` collection artifacts assume this version for the installation process. This version can be downloaded from [Oracle edelivery website](#) or [Oracle Technology Network \(OTN\)](#).
3. The `power_aix_oracle` collection uses the [power\\_aix collection](#) for some of the configuration requirements, so it must be installed on the Ansible server.

**Note:** If the LPAR has not been bootstrapped for Ansible before, it is a best practice to do so now. To bootstrap an LPAR for Ansible, complete the following steps:

1. Help ensure that the `ibm.power_aix` collection is installed.
2. Copy `demo_bootstrap.yml` from the `playbooks` directory of the collection to your workstation.
3. Run that playbook against your LPAR to bootstrap it by running the following command:  

```
ansible-playbook demo_bootstrap.yml -l mylpar.
```

## Installing the power\_aix\_oracle collection

Use the command that is in Example 4-2 on page 199 to download and install the collection.

<sup>1</sup> Source: [https://github.com/IBM/ansible-power-aix-oracle/blob/main/docs/README\\_ORA\\_SI\\_Play.pdf](https://github.com/IBM/ansible-power-aix-oracle/blob/main/docs/README_ORA_SI_Play.pdf)

*Example 4-2 Installing the `ibm.power_aix_oracle` collection from the Ansible Galaxy website*

```
ansible-galaxy collection install ibm.power_aix_oracle
Process install dependency map
Starting collection install process
Installing 'ibm.power_aix_oracle:1.1.1' to
'/root/.ansible/collections/ansible_collections/ibm/power_aix_oracle'
```

After the collection is installed successfully, all components, such as roles, variables, and playbooks, are saved in the directory that is specified in the last output line of Example 4-2. You may copy that directory to a working directory to update it with your variables and other parameters to avoid changing the installed source.

Figure 4-3 show the commands to copy the collection and the contents of the collection.

```
root@ansible ~# pwd
/root
root@ansible ~# mkdir workspace
root@ansible ~# cp -r /root/.ansible/collections/ansible_collections/ibm/power_aix_oracle/ workspace/
root@ansible ~# cd workspace/power_aix_oracle/; ls
CODE_OF_CONDUCT.md  LICENSE          README.md        collections      inventory        roles
CONTRIBUTING.md    MAINTAINERS.md  _config.yml     demo_play_aix_oracle.yml  meta            tests
FILES.json          MANIFEST.json   ansible.cfg      docs             pics            vars
root@ansible power_aix_oracle#
```

*Figure 4-3 Copying the collection's directory into a working directory and showing its contents*

The roles in the collection do much work. There are three key roles that are stored in the roles directory, as shown in Figure 4-3.

### The `power_aix_oracle` collection roles

There are three roles that are used in the `power_aix_oracle` collection:

► The `preconfig` role

The `preconfig` role performs AIX configuration tasks that are needed for Oracle installation. It checks whether the requirements are met, and if they are not, it attempts to remedy the situation. It checks the following items:

- The main OS (rootvg) file systems meet the minimum requirements.
- The AIX file sets that are required by Oracle database, such as `bos.adt.*`, `bos.perf.*`, `rsct.*`, `X1C.*`, and others.
- Red Hat Package Manager packages for tools that are used during the installation.
- The network configuration, such as `hostname`, `real_hostname`, corresponding IP address, and DNS server definition.
- Validate the disks to use for the Oracle database installation, depending on whether it is a file system-based or ASM-based installation.
- AIX tuning, such as `maxuproc`, a Red Hat OpenShift Container Platform device, and paging space.

Restart to enact the modified configurations.

► The `oracle_install` role

This role performs Oracle binary file installation. It sets the necessary components based on whether it is a file system or ASM installation, copies the binary files over to the AIX LPAR, and installs them.

The role follows this sequence:

- a. Sets the execution variables, setting `grid_asm_flag` to `true` for an ASM-based installation or to `false` for a file system-based installation.
- b. Creates the OS-level group and user that are necessary for the Oracle database installation (that is, the `oper` group and the `oinstall` user).

For file system-based installation, it complete the following steps:

- i. Creates a volume group on the database disks.
- ii. Creates and mounts the file systems at the correct sizes in the volume group.

For an ASM-based installation, it completes the following steps:

- i. Creates the Oracle grid home directory.
- ii. Sets the ownership and access mode correctly for the ASM disks.
- iii. Sets up the grid source files and `rootpre` script.
- iv. Generates the grid response file and uses it for the grid installation.
- v. Runs the `orainstallroot` and `root` scripts.
- vi. Runs the grid `ConfigTools` script.

- c. Perform the Oracle database installation by using the following steps:

- i. Sets up the `oinstall` user profile by pointing to the correct `tmp` directory and defining the oracle SID and Oracle home directory.
- ii. Copies the Oracle installation binary files over from the repository to the LPAR.
- iii. Generates the Oracle installation response file and uses it to install the binary files.

► `oracle_createdb` role

This role is used to create database instances in the database server. It identifies whether the database server uses ASM storage or file system storage and runs the corresponding routine to create the database instance. It also uses variables such as target instance SID, password, and character set, which are defined in the variables during creation.

To create an oracle database instance on a server that uses ASM storage, complete the following steps:

- a. Help ensure that no database instance exists with the same target SID.
- b. Generate the database instance creation template or script.
- c. Use the template or script to create the database instance.

To create an Oracle database instance on a server that uses file system storage, complete the following steps:

- a. Help ensure that no database instance exists with the same target SID.
- b. Create a volume group for the database instance by using an available volume set for the database.
- c. Create and mount a file system for the database instance on the volume group and help ensure its ownership and access mode.
- d. Generate the database instance creation template or script.
- e. Use the template or script to create the database instance.



## Installing Oracle DB 19c on AIX and creating a database instance

To install Oracle DB 19c on AIX and create a database instance, complete the following steps:

1. Help ensure that all prerequisites that are documented in “Single-node Oracle database installation prerequisites” on page 198 are met, including setting up the AIX LPAR, installing the `ibm.power_aix` collection, and downloading the Oracle database installation software.
2. Set up the AIX LPAR as an Ansible client by adding it to the inventory file and exchanging the Ansible server’s Secure Shell (SSH) key with it.
3. Help ensure that the `ibm.power_aix_oracle` collection is installed, as shown in Example 4-2 on page 199.
4. Optionally, copy the `ibm.power_aix_oracle` directory to a temp working directory, as shown in Figure 4-3 on page 199.
5. Copy `netshvc.conf` and `resolv.conf` from the `/etc` directory of the AIX LPAR into the `roles/preconfig/files/` directory that is inside the `workspace/power_aix_oracle/roles/preconfig/files` directory, where `workspace` is the temporary working directory that you copied the collection directory to in step 4.
6. Modify the Oracle binary files location path variable `<oracledbaix19c>` in the file `workspace/power_aix_oracle/vars/oracle_params.yml` and set the `grid_asm_flag` flag value to `true` for an ASM-based installation or `false` for a file system-based installation (the default option is file system-based). Go through all other parameters in that `oracle_params.yml` variables file and modify it for your environment.
7. Run the playbook `demo_play_aix_oracle.yml`, which includes the variables file. The playbook runs the three roles sequentially.

Example 4-3 shows a playbook (`ansible-playbook demo_play_aix_oracle.yml`) that installs the Oracle database and creates a database instance.

*Example 4-3 A playbook that installs Oracle DB and creates an instance based on the vars file*

---

```
- hosts: all
  gather_facts: yes
  vars_files: vars/oracle_params.yml
  roles:
    - role: preconfig
      tags: preconfig
    - role: oracle_install
      tags: oracle_install
    - role: oracle_createdb
      tags: oracle_createdb
```

---

To check on future updates of the collection, see its [documentation site](#).

### 4.6.3 Automating the deployment of Oracle RAC with Ansible

With Oracle RAC, customers can run a single Oracle Database across multiple servers to maximize availability and enable horizontal scalability, while accessing shared storage.<sup>2</sup> IBM AIX is a commonly used platform for hosting Oracle RAC.

---

<sup>2</sup> Source: <https://www.oracle.com/qa/database/real-application-clusters/>

Figure 4-4 shows a high-level overview of an Oracle RAC installation on an existing infrastructure.

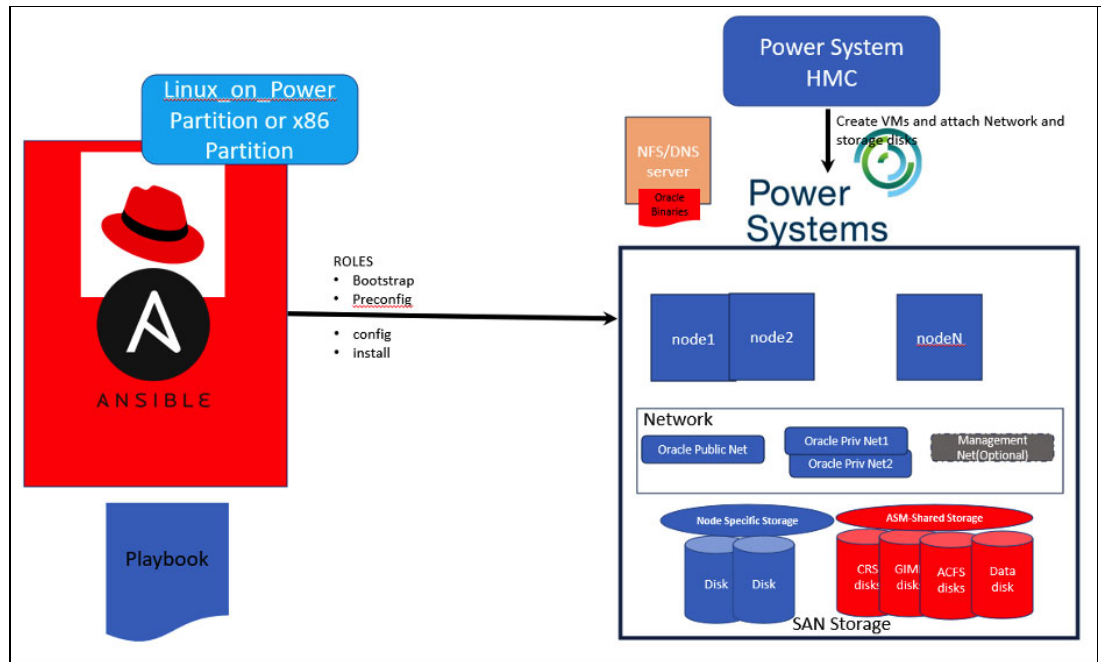


Figure 4-4 Oracle RAC deployment topology on existing infrastructure<sup>3</sup>

Figure 4-5 shows the same overview for both infrastructure provisioning and Oracle RAC software installation automation.

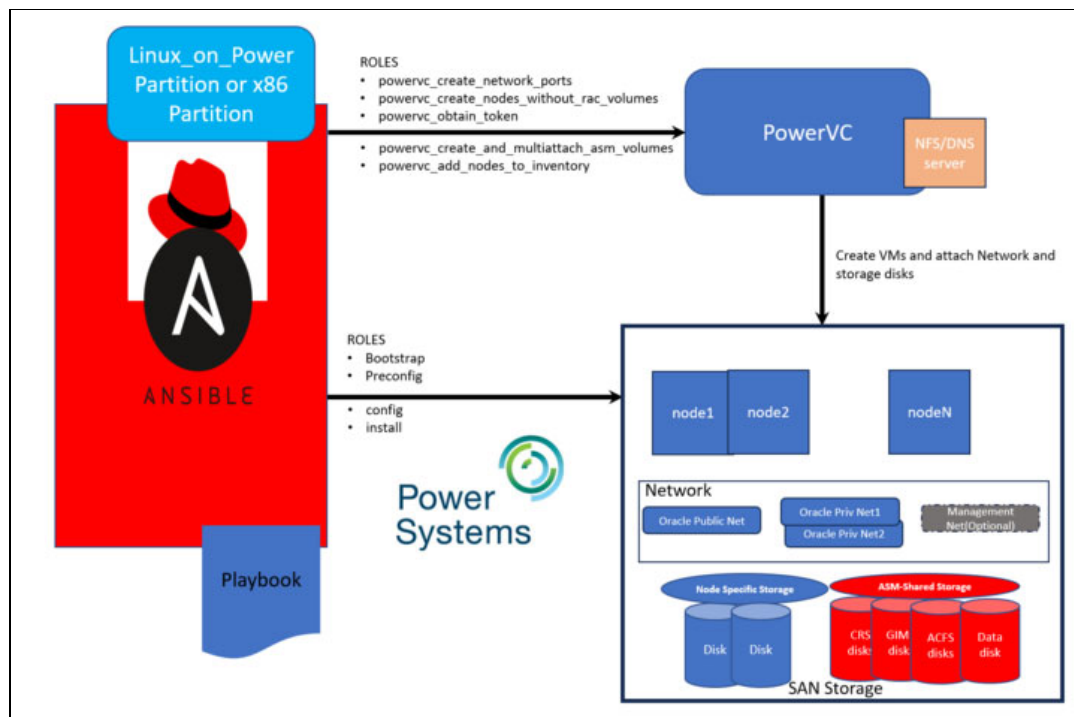


Figure 4-5 Oracle RAC deployment topology with IBM PowerVC automating the infrastructure layer<sup>4</sup>

<sup>3</sup> Source:

[https://github.com/IBM/power-aix-oracle-rac-asm/blob/main/docs/README\\_ORACLE\\_RAC\\_PLAYBOOK\\_V1.3.pdf](https://github.com/IBM/power-aix-oracle-rac-asm/blob/main/docs/README_ORACLE_RAC_PLAYBOOK_V1.3.pdf)

Oracle RAC installation, on AIX and elsewhere, offers has many complexities both at the infrastructure layer setup and in the software installation requirements. At the infrastructure layer, the complexities encompass setting up the AIX nodes on hosts that meet the RAC requirements. These requirements include setting kernel tunable parameters, setting network attributes, setting shared disks attributes, and setting up ssh password-less access among others. It is a tedious, repetitive, and error-prone process when done manually. Similarly, the manual process of the grid and database software installation is interactive and requires a user's attentive physical presence for hours.

The Ansible Oracle RAC collection (`ibm.power_aix_oracle_rac_asm`) that is available at [Ansible Galaxy](#) and [GitHub](#) simplifies the installation of Oracle RAC 19c on the AIX OS running on IBM Power servers by automating both the infrastructure setup operation and the software installation and configuration operation. It contains playbooks and many supporting roles and other artifacts that automate both layers.

The infrastructure layer automation that the collection provides requires IBM PowerVC. If your environment is not equipped with PowerVC, you can prepare the infrastructure manually and then use the collection for the Oracle RAC software installation. Otherwise, you can use the collection to automate both layers of the process.

## Setting up Ansible Server for Oracle RAC installation automation

This section covers Ansible server requirements for installing the Oracle RAC collection along with more Ansible server configurations that are required for that collection.

### Ansible server requirements for Oracle RAC collection

The Ansible server must meet the following requirements to be used for Oracle RAC collection:

- ▶ The Ansible server must be Version 2.9 or later.
- ▶ The `python3-netaddr`, `perl`, `expect`, and `wget` Red Hat Package Managers must be installed.
- ▶ The `ibm.power_aix` collection (see 1.5.4, “Ansible for AIX” on page 40) must be installed because the Oracle RAC collection uses some of its modules in the setup process.
- ▶ The `ansible.utils` collection is required for the Oracle RAC collection to work. If it is not installed, then use the `ansible-galaxy install ansible.utils` command to install it.
- ▶ The [OpenStack SDK](#) must be installed when automating the provisioning of the infrastructure layer.

**Troubleshooting note:** There was a scenario where password-less authentication between the nodes failed with security-like errors that pointed to a missing `python3-selinux` Red Hat Package Manager on the Ansible server. Disabling SELinux in the Ansible server resolved that issue. This approach provides a workaround until a permanent resolution is reached.

---

<sup>4</sup> Source:

[https://github.com/IBM/power-aix-oracle-rac-asm/blob/main/docs/README\\_ORACLE\\_RAC\\_PLAYBOOK\\_V1.3.pdf](https://github.com/IBM/power-aix-oracle-rac-asm/blob/main/docs/README_ORACLE_RAC_PLAYBOOK_V1.3.pdf)

## Installing the power\_aix\_oracle\_rac\_asm collection

To download and install the collection, use the example that is provided in Example 4-4. If your Ansible server does not have access to the internet, then download the collection's .tar file to your workstation, transfer it to the Ansible server, and then run the command with the same syntax but point to the .tar file instead.

*Example 4-4 Installing the power\_aix\_oracle\_rac\_asm collection*

```
ansible-galaxy collection install ibm.power_aix_oracle_rac_asm
Process install dependency map
Starting collection install process
Installing 'ibm.power_aix_oracle_rac_asm:1.2.1' to
'/root/.ansible/collections/ansible_collections/ibm/power_aix_oracle_rac_asm'
```

After the collection is installed successfully, all its contents are stored in the directory that is shown in the last line of the Example 4-4 output. Copy that directory to your workspace and work with it. This way, the original collection's directory remains as an unchanged reference.

Figure 4-6 shows contents of the collection's directory after copying it to the workspace.

```
root@ansible workspace# pwd
/root/workspace
root@ansible workspace# cp -r ~/.ansible/collections/ansible_collections/ibm/power_aix_oracle_rac_asm/ .
root@ansible workspace# cd power_aix_oracle_rac_asm; ls
CODE_OF_CONDUCT.md  _config.yml          inventory
CONTRIBUTING.md   ansible-navigator.yml meta
MAINTAINERS.md     ansible.cfg           powervc_build_AIX_RAC_nodes.yml
MANIFEST.json      collections           roles
README.md          install_and_configure_Oracle_RAC.yml vars
root@ansible power_aix_oracle_rac_asm#
```

*Figure 4-6 Copying the Oracle RAC installation collection directory to the workspace and showing its contents*

The files `powervc_build_AIX_RAC_nodes.yml` and `install_and_configure_Oracle_RAC.yml` are the playbooks that are used for automating the infrastructure layer and the Oracle RAC installation. The `roles` directory hosts the roles supporting those playbooks, and the `vars` directory contains the files where the variables that are used by these playbooks and roles are set.

## Automating the infrastructure layer provisioning with PowerVC

The infrastructure layer automation involves the following activities, which are automated through the `powervc_build_AIX_RAC_nodes.yml` playbook and its supportive Ansible roles that are provided in the collection:

- ▶ An operational PowerVC 2.1.1 server or later.
- ▶ Setting up a PowerVC image that is configured with Oracle RAC specifications.
- ▶ Setting up Oracle RAC required networks.
- ▶ Setting up the ASM storage volumes.

## Setting up the PowerVC image for Oracle RAC

The PowerVC AIX image should follow these specifications:

- ▶ It should have a rootvg disk size of at least 50 GB. The larger size is to accommodate a 16 GB paging space and 8 GB for /tmp to be used as the Ansible temp directory. This rootvg disk is hdisk0 in each node.
- ▶ A second disk of 75 GB that is not assigned to a volume group. It is used by the installer to create the oravg volume group, which hosts the Oracle Grid HOME directory. The disk is hdisk1 in the nodes. The `<*ofa_fs>` variable in the `vars/powervc_rac.yml` file is set to 73G because the disk size is 75 GB. If you capture an image with a larger sized disk, then you may want to update the value of that variable.

**Note:** Although the hdisks in the first two bullets are deployed by PowerVC as hdisk0 and hdisk1, the requirement is that each of them has the same hdisk number in both nodes. However, they do not have to be hdisk0 and hdisk1 respectively.

- ▶ The OS that is installed on the image should be AIX 7.2 TL4 SP1 or later or AIX 7.3.
- ▶ The following file sets must be installed on the AIX version before you install the Oracle RAC software. Although they may be installed in the nodes after they are created, the process becomes simpler if they are installed in the source LPAR before capturing it as the PowerVC image.
  - bos.adt.base
  - bos.adt.lib
  - bos.adt.libm
  - bos.perf.libperfstat
  - bos.perf.perfstat
  - bos.perf.proctools
  - bos.loc.utf.EN\_US
  - bos.rte.security
  - bos.rte.bind\_cmds
  - bos.compat.libs
  - xlc.aix61.rte
  - xlc.rte
  - rsct.basic.rte
  - rsct.compat.clients.rte
  - xlsmp.msg.EN\_US.rte
  - xlf rte.aix61
  - openssh.base.client
  - expect.base
  - perl.rte
  - Java8\_64.jre
  - dsm
- ▶ Extract the Red Hat Package Manager and install it on the image. You can download the latest version from [here](#).
- ▶ Update the image section in the `vars/powervc.yml` file in the collection with the `image`, `image_aix_version`, and `image_password`, with the latter set to the AIX root password value.

## Setting up networks for Oracle RAC

A 2-node Oracle RAC cluster has the following network requirements:

- ▶ The two nodes require nine IP addresses: four IP addresses per node and one for the RAC scan service. These nine IP addresses must be created across three different networks.

**Note:** Although the Oracle RAC collection can support up to an 8-node cluster, it has been extensively tested for a 2-node cluster. If you intend to use the collection to provision a cluster with more than two nodes, then all variables files must be reviewed to update the variables' values.

Also, this collection of documents uses a single IP address for RAC scan service, but it is for testing purpose only. Oracle recommends that you use three IP addresses for this service.

- ▶ The four IP addresses per node are as follows:
  - One public IP address for the node's external access, which is sourced from the first network.
  - Two private IP addresses, which are sourced from the second and third networks for Oracle RAC private interconnect.
  - One virtual IP address per node.
- ▶ Five of these nine IP addresses are sourced from network 1 and must support a 2-way resolution against a DNS server.
- ▶ Network one must be accessible from the Ansible server because the public IP addresses or their hostnames are used as Ansible inventory entries.
- ▶ All three networks must be routed through the Virtual I/O Server (VIOS) and added to PowerVC.
- ▶ All network interfaces must be consistent across the nodes. For example, if the public IP address for node 1 is set to en0, then node2 must also use en0 for its public IP address. Table 4-1 details the requirements for the IP addresses on a 2-node cluster.

**Note:** When using PowerVC for provisioning the nodes, help ensure that both <nodeX\_net\_ports> variables in the vars/powervc.yml file list the public port, private 1, and virtual 2 ports in this sequence, which helps ensure that you use en0, en1, and en2.

Table 4-1 IP addresses specifications for a 2-node Oracle RAC

Location	Function	Network #	Type (Interface)	DNS required?
Node1	Public IP	Network 1	Physical (en0)	Yes
Node2	Public IP	Network 1	Physical (en0)	Yes
Node1	First private IP	Network 2	Physical (en1)	No
Node2	First private IP	Network 2	Physical (en1)	No
Node1	Second private IP	Network 3	Physical (en2)	No
Node2	Second private IP	Network 3	Physical (en2)	No
Node1	Virtual IP	Network 1	Virtual (N/A)	Yes

Location	Function	Network #	Type (Interface)	DNS required?
Node2	Virtual IP	Network 1	Virtual (N/A)	Yes
Cluster	Cluster scan IP	Network 1	Virtual (N/A)	Yes

- ▶ Update the network section of the `vars/powervc.yml` file, supplying the network names and IP addresses in the variables, which are named according to the functions that are defined in Table 4-1 on page 206.
- ▶ Update the `vars/powervc.yml` file with the following additional variables:
  - DNS server and domain.  
The DNS server should be capable of forward and reverse name resolution for all five IP addresses that are labeled Yes in the last column of Table 4-1 on page 206.
  - NTP server.  
The name server is needed to keep the cluster operational. Alternatively, you can help ensure that the date and time are synchronized between the two nodes.
  - NFS server and its export directory, and the directory to use as an NFS mount point in the nodes.  
Oracle RAC installation binary files, including the grid, database, OPatch, and RU, should be stored in subdirectories under that export directory in the NFS server. Check whether a node that is deployed from the PowerVC image can successfully mount the export directory from the NFS server.

### ***Defining the shared storage for Oracle RAC***

Oracle RAC uses ASM disks that are shared among the RAC nodes. The RAC collection creates four ASM diskgroups by using ASM shared disks. Each diskgroup can have one or more disks.

Disks of each disk group should have the same size, characteristics, and similar I/O speed. Here are the four diskgroups:

- ▶ The OCRVOTE disk group stores Oracle Cluster Registry (OCR) and voting disks information.
- ▶ The Grid Infrastructure Management Recovery (GIMR) disk group contains a multi-tenant database (Management Database (MGMTDB)) with one pluggable database.
- ▶ The ASM Cluster File System (ACFS) disk group is used for staging the Oracle database home binary file.
- ▶ The DATA disk group is used for staging the database files.

The Oracle RAC installer expects each disk to have the same hdisk number in all RAC nodes.

**Note:** When using the PowerVC playbook from the collection to provision the nodes, it helps ensure that each node has the same hdisk number across all nodes. It does so by creating them one at a time, and after creating each one, it attaches it to all nodes, and then runs `cfgmgr` to help ensure that it captured the next available sequential number for the hdisk in all nodes before moving on to the next disk.

Table 4-2 cross matches the disk groups and their corresponding variable names, disks count, and each disk's size as set as default in the disks list of the vars/powervc.yml file. It also shows the corresponding hdisk number as set in the <diskgroups> variable in the vars/powervc\_rac.yml file.

Table 4-2 Diskgroups names and corresponding variables in vars/powervc.yml

Diskgroup	Variable name	Disks count	The hdisk number	Disk size
OCRVOTE	<“{{racName}}-ASMOCRx”>	4	2 3 4 5	10
GIMR	<“{{racName}}-GIMRx”>	2	6 7	40
ACFS	<“{{racName}}-ACFS-DBHome”>	1	8	75
DATA	<“{{racName}}-DBDiskx”>	2	9 10	10

Consider the following items as you update these variables in their variables file:

- ▶ The <diskgroups> variable in the vars/powervc\_rac.yml file lists the hdisk number of these disks. Within a PowerVC nodes deployment, the image uses hdisk0 and hdisk1, as described in “Setting up the PowerVC image for Oracle RAC” on page 205. The hdisk number of each of these ASM disks are as shown in the ‘hdisk number’ column in Table 4-2. If you change the count of any of the diskgroups, update the hdisk numbers in the <diskgroups> variable in the vars/powervc\_rac.yml file.
- ▶ You may change the disks’ sizes to meet your requirements. Help ensure that all disks of a given diskgroup are the same size.
- ▶ The vol\_size\_GB variable in vars/powervc\_rac.yml is set to 75 GB based on the ACFS disk size of 75 GB. If you change that disk’s size in vars/powervc.yml, then update that variable in the vars/powervc\_rac.yml file too.

**Note:** When using a PowerVC playbook from the collection to provision the nodes, it helps ensure that all non-rootvg disks’ headers are clear and have no PVIDs because it creates them from the storage subsystem.

For manual setup of the nodes, use `chdev -l hdiskX -a pv=clear` to clear the PVID. Then, use `lquerypv -h /dev/hdiskX` to check whether the header is clear, If it is not, then use `dd if=/dev/zero of=/dev/hdiskX bs=1024k count=100` to clear it.

### **The collection roles that are used for infrastructure provisioning**

The power\_aix\_oracle\_rac\_asm collection contains the following roles pertaining to the infrastructure layer provisioning:

- ▶ `powervc_create_network_ports`  
This role creates an OpenStack port to use during the node creation. A port defines the IP address and network to use for an interface.
- ▶ `powervc_create_nodes_without_rac_volumes`  
This role uses parameters that are set in the vars/powervc.yml file to create the cluster nodes.
- ▶ `powervc_obtain_token`  
This role obtains a PowerVC access token to establish a Representational State Transfer (REST) application programming interface (API) connection from an Ansible server to a PowerVC server. The subsequent ASM disks creation role requires REST API access, hence the need for the token.



- ▶ `powervc_create_and_multiattach_asm_volumes`  
This role creates the ASM disks one at a time. On creating each disk, this role attaches it to all nodes and runs `cfgmgr` to help ensure that the disk maintains the same `hdisk` number in all nodes as required by the Oracle RAC installer.
- ▶ `powervc_add_nodes_to_inventory`  
This role updates the inventory file with the nodes and more parameters to set it up for Ansible management, and then prepares the environment for execution of the second playbook, which is responsible for grid and database software installation.

## **Automating the Oracle RAC software installation**

Oracle RAC software installation automation involves activities that are automated by the `install_and_configure_oracle_rac.yml` playbook and its supported Ansible roles, which are provided by the collection. Oracle database installation by using the collection assumes that the `software-installation` only option is on the ACFS shared file system.

When you build the infrastructure manually, installing the grid and database software requires that you update the `vars/rac.yml` file with the values of the required variables and include only the `vars/rac.yml` file in the playbook. If you automate the infrastructure provisioning with PowerVC, then you populate those variables in the `vars/powervc.yml` file, which are then referenced in the `vars/powervc_rac.yml` file. If so, include both the `vars/powervc.yml` file and the `vars/powervc_rac.yml` file in the playbook.

### ***The collection roles that are used for the Oracle RAC software installation***

The `power_aix_oracle_rac_asm` collection contains the following roles pertaining to the grid and database software installation:

- ▶ `bootstrap`  
This role sets up the basic environment to enable the full function of Ansible to set the name server, binding, and password-less connections to the RAC nodes.
- ▶ `preconfig`  
This role sets up a basic environment with time of day, configuration for accessing the internet, and helping ensure consistent AIX versions, releases, TLs, and SsP. It also NFS mounts and installs the AIX file sets.
- ▶ `config`  
This role sets up AIX to meet the requirements for installing Oracle RAC software.
- ▶ `install`  
This role creates the ASM disk groups and ACFS, prepares for installing the grid and database, and installs `GRID_HOME` on a JFS2 file system and database `ORACLE_HOME` on the ACFS shared file system.

### ***Installation steps for the different options***

There are two options for using the collection to install Oracle RAC on AIX:

- ▶ Option 1: Manually configure the infrastructure while automating the software installation.
- ▶ Option 2: Fully automate both operations through a single playbook run.

Option 1 is shown in Figure 4-4 on page 202. Option 2 is shown in Figure 4-5 on page 202

### **Option 1 installation steps**

Installation option 1 entails preparing the cluster nodes manually and then using the collection to install the RAC software. Complete the following steps:

1. Create the Oracle RAC cluster AIX nodes manually.
2. Create the local and shared storage volumes manually per the specifications and attach them to the nodes.
3. Help ensure that the nodes are configured with the correct networks per the specifications.
4. Help ensure that the hostnames, and virtual and scan IP addresses are added correctly to the DNS server.
5. If no NTP server is configured for the nodes, help ensure that the clocks on the cluster nodes are in sync.
6. NFS servers are needed for the AIX file sets installation and for staging the Oracle 19c software.
7. Update the vars/rac.yml file to reflect the correct values for all the concerned variables.
8. Review the inventory and ansible.cfg files and help ensure that the nodes are added correctly to the inventory file with the correct name of the nodes and the group containing them.
9. Update the install\_and\_configure\_Oracle\_RAC.yml playbook, as shown in Example 4-5 and in the following steps:
  - a. Uncomment the hosts: line and set the field by specifying the inventory group name, as described in step 8.
  - b. Uncomment the first variables file (named vars/rac.yml) to include its variables in this run.
10. Run the playbook by using the following command:

```
ansible-playbook install_and_configure_Oracle_RAC.yml
```

The output is shown in Example 4-5.

---

#### *Example 4-5 Contents of the install\_and\_configure\_Oracle\_RAC.yml playbook*

---

```
---
# install_and_configure_Oracle_RAC.yml
- hosts: "{{ racName }}"
  gather_facts: no
  vars_files:
#   - vars/rac.yml
#   - vars/powervc.yml
#   - vars/powervc_rac.yml
  roles:
    - role: bootstrap
      vars:
        download_dir: "~"
        target_dir: "/tmp/.ansible.cpidir"
      tags: bootstrap
    - role: preconfig
      tags: preconfig
    - role: config
      tags: config
    - role: install
      tags: install
```

---

## **Option 2 installation assumptions**

The following assumptions apply to installation option 2.

A variable that is named `<racName>` is used throughout the automation process. This variable is required as an extra parameter for playbooks of both operations. The following parameters inherit the `<racName>` variable:

- ▶ Node names and hostnames append a counter to the `<racName>`. So, if the `<racName>` is set to `myorac`, then the nodes are named `myorac1` and `myorac2`.
- ▶ The nodes group on the Ansible inventory file is named after the `<racName>`.
- ▶ The target hosts in the software installation playbook are set to the Ansible inventory group, which is named `<racName>`.
- ▶ The hostname of the cluster scan IP address appends `'-scan'` to the `<racName>`.
- ▶ The hostnames of all node IP addresses are based on the nodes names, which are based on the `<racName>`.

**Troubleshooting note:** If any of the nodes are rebuilt (or their ssh identity changes) after they are added to the Ansible server `~/ .ssh/known_hosts` file, then entries in that file must be clear before there is any subsequent attempt to install Oracle RAC software in these nodes. This situation is true for installation option 1 too.

This precautionary step prevents a “WARNING: POSSIBLE DNS SPOOFING DETECTED!” error, which causes password-less ssh connection and playbook execution on node1 failures, and an incomplete software installation process.

## **Option 2 installation steps**

Installation option 2 entails using the collection to automate both preparing the infrastructure layer (operation 1) and installing the RAC software (operation 2) in a single run. The process imports the playbook for operation 2 as the last step of operation 1’s playbook to automate the processes sequentially.

Complete the following steps:

1. Create DNS entries in the DNS server for the five addresses that must have DNS entries per Table 4-1 on page 206.
2. Configure the NFS server and store the Oracle RAC software in subdirectories in its export directory. The export directory value must be set in the `vars/powervc.yml` file, and the names of the binary files and their hosting subdirectories must be set in the `vars/powervc_rac.yml` file.
3. Update the `vars/powervc.yml` file with the values for all images, networks, and shared storage disk variables, the DNS, NFS, and NTP servers, and the PowerVC server credentials.
4. Update the `vars/powervc_rac.yml` file for the `hdisk0` with the ACFS disk size, as described in “Option 1 installation steps” on page 210. Update the grid and Oracle passwords and any other variables that you deem necessary for your environment.
5. Copy the PowerVC certificate file from the `/etc/pki/tls/certs/powervc.crt` file in the PowerVC server to the Ansible server.
6. Copy the `/opt/ibm/powervc/powervcrc` file from the PowerVC server to the Ansible server, and update its `OS_CACERT` with the location where you copied the PowerVC certificate file. Update `OS_CACERT` with the user ID and password of the PowerVC server and source it.

Example 4-6 shows the playbook for the infrastructure provisioning layer.

*Example 4-6 Contents of the powervc\_build\_AIX\_RAC\_nodes.yml playbook*

---

```
---
# powervc_build_AIX_RAC_nodes.yml

- name: Build and configure the RAC nodes by using PowerVC
  hosts: localhost
  gather_facts: no
  vars_files:
    - vars/powervc.yml
  tasks:
    - include_vars: "vars/powervc.yml"
    - fail:
        msg: "racName is required for this playbook to build a dual-node Oracle
        RAC."
        when: racName is not defined

    - name: Display the input name prepatch and count of VMs to be built
      debug:
        msg: "Creating nodes {{racName}}1 and {{racName}}2 for this dual-node Oracle
        RAC."

    - name: define the network ports based on the networks and IP addresses to be
      used.
      import_role: name=powervc_create_network_ports

    - name: Create new AIX VMs to act as Oracle RAC nodes
      import_role: name=powervc_create_nodes_without_rac_volumes

    - import_role: name=powervc_obtain_token
    - include_role: name=powervc_create_and_multiattach_asm_volumes
      with_items: "{{ disks }}"

    - name: Now, the nodes are good to go, add them to the inventory file to be
      managed by Ansible
      import_role: name=powervc_add_nodes_to_inventory

# Importing the playbook to be used for installing and configuring the Oracle RAC.
- import_playbook: install_and_configure_Oracle_RAC.yml
```

---

7. The last step in Example 4-6 invokes the software installation playbook that is shown in Example 4-7.

Update the software installation playbook by uncommenting the last variables file (vars/powervc\_rac.yml) because the variables that are defined in that file are needed in the playbook.

*Example 4-7 Contents of install\_and\_configure\_Oracle\_RAC.yml*

---

```
---
# install_and_configure_Oracle_RAC.yml
# PowerVC based deployments use variable files
vars/powervc.yml,vars/powervc_rac.yml
```

```

# If the LPARs are build manually to automate oracle RAC deployment use variable
file vars/rac.yml
#- hosts: "{{ racName }}" # racName variable is defined when you use the PowerVC
automation scripts for building the AIX LPARs
#- hosts: orac # Get the group name from inventory file, which
contains the oracle cluster nodes
gather_facts: no
vars_files:
# - vars/powervc.yml
# - vars/powervc_rac.yml
# - vars/rac.yml
roles:
- role: bootstrap
vars:
download_dir: "~"
target_dir: "/tmp/.ansible.cpubdir"
tags: bootstrap
- role: preconfig
tags: preconfig
- role: config
tags: config
- role: install
tags: install

```

---

8. Run the operation 1 playbook (which performs both operations) by running the following command:

```
ansible-playbook powervc_build_AIX_RAC_nodes.yml -e racName=myorac.
```

### ***Running the playbooks separately***

You may choose to run the two playbooks in two separate steps for different reasons, for example:

- ▶ A time gap is needed to prepare the DNS server and update it with the required addresses.
- ▶ A time gap is needed to prepare the NFS server or to set it up with the required Oracle software binary files to use in the second operation.
- ▶ You want to interrogate the nodes for all prerequisites to help ensure that the first operation playbook satisfied them.

If so, you can run the infrastructure playbook as described in Example 4-6 on page 212, except that you comment out the last line in that playbook (shown in Example 4-7 on page 212) to not import the software installation playbook.

When you are ready to do the software installation, complete the following steps:

1. Uncomment the middle variables file (vars/powervc.yml) in the software installation playbook (shown in Example 4-7 on page 212) so that both the second and third variable files with PowerVC in their name are active.
2. Run the software installation playbook with <racName>:

```
ansible-playbook install_and_configure_Oracle_RAC.yml -e racName=myorac
```

As you continue to work with the collection, you might find it useful to see the [collection documentation](#) for newer release updates. Furthermore, the collection's [GitHub issues page](#) is a good tool for resolving any issues.

**Note:** Oracle RAC automation collection stops with the software installation. To create database instances and manage them, see 4.6.4, “Automating Oracle DBA operations” on page 214.

## 4.6.4 Automating Oracle DBA operations

Oracle DBA tasks are a set of support activities that are done by DBAs. These tasks are mostly iterative and require attention, especially when there are many databases. Automating these tasks can help DBAs save time and avoid human errors.

The `ibm.power_aix_oracle_dba` Ansible collection is based on an open-source project that is called [Oravirt](#), which automates Oracle DBA tasks. These roles and modules were evaluated and tested to work with Oracle databases running on AIX, and they offer many playbooks.

### Prerequisites

The following software must be installed on the Ansible Controller host:

- ▶ Python 3.6 or later (Python can be installed by running `dnf install python3`).
- ▶ `Cx_oracle`, which is a Python module that makes the connection to the database by using sys privileges. For more information about `Cx_oracle`, see [Introduction to cx\\_Oracle](#).

To install `Cx_oracle` online, use one of the following commands:

- As root: `python -m pip install cx_Oracle --upgrade`
- As a non-root user: `python -m pip install cx_Oracle--upgrade --user`

For an offline installation, complete the following steps:

- a. Download the source distribution from [Pypi](#) and place it in a location, for example, `/tmp`.
- b. Run the following command:

```
python3 -m pip install --no-build-isolation /tmp/cx_Oracle-8.3.0.tar.gz
```

**Note:** If there are multiple Python versions, the Python version that was used to install `Cx_oracle` must be used for running the playbooks. To verify your Python version, see [Example 4-8](#).

[Example 4-8](#) shows how to validate which version of Python was used to install the `Cx_oracle` package and which must be used to run the playbooks.

*Example 4-8 Determining the version of Python that was used to install the `Cx_oracle` package*

---

```
$ pip3.9 show cx_Oracle
Name: cx-Oracle
Version: 8.3.0
Summary: Python interface to Oracle
Home-page: https://oracle.github.io/python-cx_Oracle
Author: "Anthony Tuininga",
Author-email: "anthony.tuininga@gmail.com",
License: BSD License
Location: /home/ansible/local/lib/python3.9/site-packages
Requires:
Required-by:
```

---

[Example 4-8](#) shows that `Cx-Oracle` is in `python3.9 site-packages`. Therefore, `python3.9` must be used as the Python interpreter to run the playbooks.

## Getting started

To use the Oracle DBA operations collection, you must have Ansible 2.9 or later on Red Hat Enterprise Linux (RHEL) 8.x or later for either Linux on Power or x86-64. You can download the collection from the following public repositories:

- ▶ [Galaxy](#)
- ▶ [GitHub](#)

To install the collection, complete the following steps:

1. Install the collection by running the following command:  

```
$ ansible-galaxy collection install ibm.power_aix_oracle_dba
```
2. Download and extract the Oracle Instant client software from [Oracle](#). At the site, click **Other Platforms**, as shown in Figure 4-7 to get the option to download the Linux on Power client.

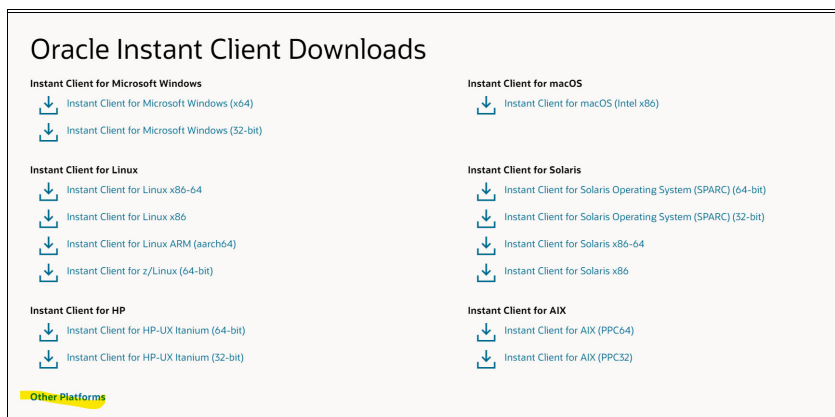


Figure 4-7 Oracle Instant Client Downloads initial page

3. Click **Linux on Power Little Endian**, as shown in Figure 4-8.

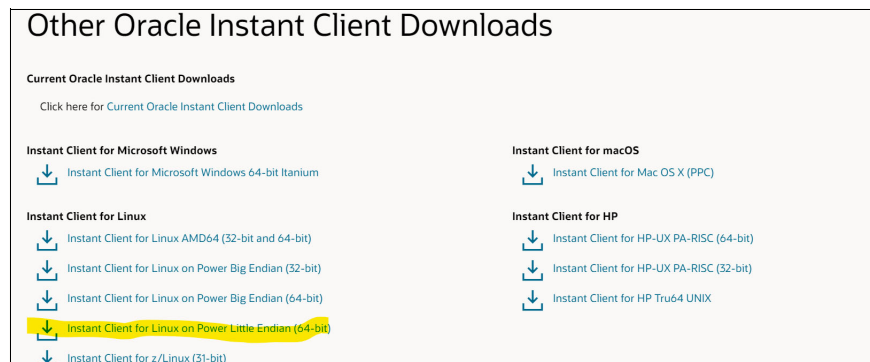


Figure 4-8 Other Oracle Instant Client Downloads

4. Install the packages `libnsl` and `libaio` by running the following commands:  

```
dnf install libnsl -y  
dnf install libaio -y
```
5. Prepare the inventory file with the hostnames of the VMs where the Oracle databases are running.

## Running administration tasks

This section provides detailed steps about running two different admin tasks:

- ▶ Creating a database
- ▶ Managing database users

### **Creating a database**

The role `oradb_create` is used to create databases. It can be used for a Non Container Database instance or a Container Database (CDB) in a single instance or RAC. In this example, we create a RAC CDB that is called “devdb” with one PDB that is called “devpdb”.

To create the database, complete the following steps:

1. Establish passwordless SSH between the Ansible user and the Oracle Database user.
2. Define the required hostname in an inventory file to be used to run the playbook.
3. Update the following three files:
  - `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml`: Contains the SYS user password of ASM and the SYS password, which must be set to the new database.
  - `{{ collection_dir }}/power_aix_oracle_dba/playbooks/create-db.yml`: Contains the playbook that runs the `oradb_create` role.
  - `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/create-db-vars.yml`: Contains all the variables to create a database. Multiple databases can be created by providing the variables as a list.
4. Update the `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml` file with the passwords:
  - a. Go to the playbooks directory and update the file with the system password for asm and dba, as shown in Example 4-9.

#### *Example 4-9 Updating the passwords*

---

```
$ cat vault.yml
default_gipass: Oracle4u
default_dbpass: Oracle4u
```

---

- b. Encrypt the file by running the following command:

```
$ ansible-vault encrypt vault.yml
```

5. Update the hosts and `remote_user` in the directory, as shown in Example 4-10:  
`{{ collection_dir }}/power_aix_oracle_dba/playbooks/create-db.yml` file

#### *Example 4-10 Modifying the create-db.yml file*

---

```
- name: Create a Database
  hosts: rac91                # Target LPAR hostname defined in the inventory
  file.
  remote_user: oracle        # Oracle Database Username
  vars_file:
    - vars/create-db-vars.yml
    - vars/vault.yml
  roles:
    - { role: ibm.power_aix_oracle_dba.oradb_create }
```

---



6. Update the following variables, as shown in Example 4-11:

```
{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/create-db-vars.yml
```

*Example 4-11 Variables in the create-db-vars.yml file*

---

```
oracle_stage: /tmp # Location on the target AIX LPAR to stage response files.
oracle_inventory_loc: /u01/app/oraInventory
oracle_base: /u01/base
oracle_dbf_dir_asm: '+DATA1' # If storage_type=ASM this is where the database is
placed.
oracle_reco_dir_asm: '+DATA1' # If storage_type=ASM this is where the fast
recovery area is
oracle_databases: # Dictionary describing the databases to be created.
  - home: db1
    oracle_version_db: 19.3.0.0 # For a 19c database, the version should be
19.3.0.0
    oracle_home: /u01/app/19c_ansible # Oracle Home location.
    oracle_edition: EE # The edition of database-server (EE,SE,SEONE)
    oracle_db_name: devdb # Database name
    oracle_db_type: RAC # Type of database (RAC,RACONENODE,SI)
    is_container: True # (true/false) Is the database a container database.
    pdb_prepatch: devpdb # Pluggable database name.
    num_pdbs: 1 # Number of pluggable databases.
    storage_type: ASM # Database storage to be used. ASM or FS.
    service_name: db19c # Initial service to be created.
    oracle_init_params: "" # initialization parameters, comma separated
    oracle_db_mem_totalmb: 10000 # Amount of RAM to be used for SGA + pGA
    oracle_database_type: MULTIPURPOSE # MULTIPURPOSE|DATA_WAREHOUSING|OLTP
    redolog_size_in_mb: 512 # Redolog size in MB
    state: present # present | absent
```

---

7. Verify that the DB does not exist, as shown in Example 4-12.

*Example 4-12 Verifying that the database does not exist*

---

```
bash-5.1$ srvctl status database -d devdb
PRCD-1120 : The resource for database devdb could not be found.
PRCR-1001 : Resource ora.devdb.db does not exist
bash-5.1$
```

---

8. Run the command shown in Example 4-13 to run the playbook.

*Example 4-13 Output from running the playbook*

---

```
[ansible@x134vm236 playbooks]$ ansible-playbook create-db.yml -i inventory.yml
--ask-vault-pass
Vault password:
[DEPRECATION WARNING]: "include" is deprecated, use include_tasks/import_tasks
instead. This feature will be removed in version 2.16.
Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.

PLAY [Create a Database]
*****
*****
```

```
TASK [Gathering Facts]
*****
*****
[WARNING]: Platform aix on host rac93 is using the discovered Python interpreter
at /usr/bin/python3, but future installation of
another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.12/reference_appendices/interpreter_discovery.html for more information.
ok: [rac93]
```

```
TASK [oradb_create : set fact]
*****
*****
ok: [rac93] => (item={'home': 'db1', 'oracle_version_db': '19.3.0.0',
'oracle_home': '/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name':
'devdb', 'oracle_db_type': 'RAC', 'is_container': True, 'pdb_prepatch': 'devpdb',
'num_pdbs': 1, 'storage_type': 'ASM', 'service_name': 'devdb',
'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000, 'oracle_database_type':
'MULTIPURPOSE', 'redo_log_size_in_mb': 50, 'state': 'present'})
```

```
TASK [oradb_create : Create Stage directory for response file.]
*****
ok: [rac93]
```

```
TASK [oradb_create : listener | Create responsefile for listener configuration]
*****
skipping: [rac93] => (item={'home': 'db1', 'oracle_version_db': '19.3.0.0',
'oracle_home': '/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name':
'devdb', 'oracle_db_type': 'RAC', 'is_container': True, 'pdb_prepatch': 'devpdb',
'num_pdbs': 1, 'storage_type': 'ASM', 'service_name': 'devdb',
'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000, 'oracle_database_type':
'MULTIPURPOSE', 'redo_log_size_in_mb': 50, 'state': 'present'})
```

NOTE: Some output has been truncated.

```
TASK [oradb_create : Add dotprofile (2)]
*****
*****
changed: [rac93] => (item=[{'home': 'db1', 'oracle_version_db': '19.3.0.0',
'oracle_home': '/u01/app/oracle/db', 'oracle_edition': 'EE', 'oracle_db_name':
'devdb', 'oracle_db_type': 'RAC', 'is_container': True, 'pdb_prepatch': 'devpdb',
'num_pdbs': 1, 'storage_type': 'ASM', 'service_name': 'devdb',
'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000, 'oracle_database_type':
'MULTIPURPOSE', 'redo_log_size_in_mb': 50, 'state': 'present'}, {'changed': False,
'stdout': '', 'stderr': '', 'rc': 0, 'cmd': 'ps -ef | grep -w "ora_pmon_devdb"
|grep -v grep | sed \'s/^.*pmon_//g\'', 'start': '2023-09-06 02:09:57.933677',
'end': '2023-09-06 02:09:57.983788', 'delta': '0:00:00.050111', 'msg': '',
'invocation': {'module_args': {'_raw_params': 'ps -ef | grep -w "ora_pmon_devdb"
|grep -v grep | sed \'s/^.*pmon_//g\'', '_uses_shell': True, 'warn': False,
'stdin_add_newline': True, 'strip_empty_ends': True, 'argv': None, 'chdir': None,
'executable': None, 'creates': None, 'removes': None, 'stdin': None}},
'stdout_lines': [], 'stderr_lines': [], 'failed': False, 'item': {'home': 'db1',
'oracle_version_db': '19.3.0.0', 'oracle_home': '/u01/app/oracle/db',
'oracle_edition': 'EE', 'oracle_db_name': 'devdb', 'oracle_db_type': 'RAC',
'is_container': True, 'pdb_prepatch': 'devpdb', 'num_pdbs': 1, 'storage_type':
```

```
'ASM', 'service_name': 'devdb', 'oracle_init_params': '', 'oracle_db_mem_totalmb': 10000, 'oracle_database_type': 'MULTIPURPOSE', 'redo_log_size_in_mb': 50, 'state': 'present'}, 'ansible_loop_var': 'item']])
```

```
TASK [oradb_create : Check whether database is running]
```

```
*****
```

```
**
```

```
changed: [rac93]
```

```
TASK [oradb_create : debug]
```

```
*****
```

```
*****
```

```
ok: [rac93] => {
```

```
  "psout.stdout_lines": [
```

```
    "   grid 14483936          1   0 00:54:37      - 0:00 asm_pmon_+ASM1",
```

```
    "   grid 14745992          1   0 00:54:56      - 0:00 apx_pmon_+APX1",
```

```
    "  oracle 21365224         1   0 02:08:59      - 0:00 ora_pmon_devdb1"
```

```
  ]
```

```
}
```

```
PLAY RECAP
```

```
*****
```

```
*****
```

```
rac93          : ok=11   changed=4   unreachable=0   failed=0
```

```
skipped=3     rescued=0   ignored=0
```

---

9. Verify that the DB is created by running the commands that are shown in Example 4-14.

*Example 4-14 Verifying that the database was created*

---

```
bash-5.1$ srvctl status database -d devdb
Instance devdb1 is running on node rac93
Instance devdb2 is running on node rac94
```

---

### **Managing database users**

The role “oradb\_manage\_users” is used to create, drop, lock, unlock, and set expiration for database users. It uses the “oracle\_users” module. The users require privileges to access the database, which you can do by using the role “oradb\_manage\_grants”. It uses the “oracle\_grants” module.

In the following example, we create two database users (testuser1 and testuser2) in a pluggable database that is called DEVPDB that runs in a CDB and grants privileges to the users.

Complete the following steps:

1. There are two files that must be updated:

- {{ collection\_dir }} / power\_aix\_oracle\_dba / playbooks / vars / manage-users-vars.yml: Contains the database hostname, database port number, and the path to the Oracle client.
- {{ collection\_dir }} / power\_aix\_oracle\_dba / playbooks / vars / vault.yml: Contains the sys password, which is used by Cx\_oracle to connect to the database with sysdba privileges.

- Update the common variables file `{{collection_dir}}/power_aix_oracle_dba/playbooks/vars/manage-users-vars.yml`, as shown in Example 4-15.

*Example 4-15 Updating the common variables file*

---

```
# Create/Drop Database Users - Variables section
hostname: rac93 # AIX LPAR hostname where the database is
running.
listener_port: 1522 # Database port number.
oracle_db_home: /home/ansible/oracle_client # Oracle Instant Client path on
controller.
oracle_databases: # Database users list to be created
  - users:
    - schema: testuser1 # Username to be created.
      default_tablespace: users # Default table space to be assigned to the
user.
      service_name: devpdb # Database service name.
      schema_password: oracle3 # Password for the user.]
      grants_mode: enforce # enforce|append.
      grants:
        - connect # Provide the name of the privilege as
a list to grant to the user.
        - resource
          state: present # present|absent|locked|unlocked [present: Creates user,
# absent: Drops user]
# Multiple users can be created with different attributes as shown below.
- users:
  - schema: testuser2
    default_tablespace: users
    service_name: devpdb
    schema_password: oracle4
    grants_mode: enforce
    grants:
      - connect
      state: present # present|absent|locked|unlocked [present: Creates user,
# absent: drops user]
```

---

- Update the passwords file `{{ collection_dir }}/power_aix_oracle_dba/playbooks/vars/vault.yml` with the sys user password, as shown in Example 4-16. This file must be encrypted by using `ansible-vault`. While running the playbook, provide the vault password.

*Example 4-16 Updating the passwords file*

---

```
default_dbpass: Oracle4u # SYS password
default_gipass: Oracle4u # ASMSNMP password
```

---

- Encrypt the passwords file by using `ansible-vault`, as shown in Example 4-17.

*Example 4-17 Encrypting the passwords file*

---

```
$ ansible-vault encrypt vars/vault.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

---

5. Check the usernames in the database before creating them, as shown in Example 4-18.

*Example 4-18 Validating usernames in the database*

---

```
SQL> sho pdbs

          CON_ID CON_NAME                                OPEN MODE  RESTRICTED
-----
          3 DEVPDB                                     READ WRITE NO
SQL> select username from dba_users where username in ('TESTUSER1','TESTUSER2');

no rows selected
```

---

The output shows that the users do not exist.

6. Create the playbook from the {{ collection\_dir }}/power\_aix\_oracle\_dba/playbooks directory, as shown in Example 4-19.

*Example 4-19 Creating the manage-users playbook*

---

```
$ cat manage-users.yml
- hosts: localhost
  connection: local
  gather_facts: false
  vars_files:
    - vars/manage-user-vars.yml
    - vars/vault.yml
  roles:
    - { role: oradb_manage_users }
```

---

7. Run the playbook, as shown in Example 4-20.

*Example 4-20 Running the playbook to create users*

---

```
[ansible@x134vm236 playbooks]$ ansible-playbook manage-users.yml --ask-vault-pass
Vault password:

PLAY [Create DB User]
*****
*****

TASK [oradb_manage_users : Manage users (cdb/pdb)]
*****
**
changed: [localhost] => (item=port: 1522 service: devpdb schema: testuser1
state:present)
changed: [localhost] => (item=port: 1522 service: devpdb schema: testuser2
state:present)
[WARNING]: Module did not set no_log for update_password

PLAY RECAP
*****
*****
localhost          : ok=1    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

---

- Check the usernames in the database after creating them, as shown in Example 4-21, where we can see that testuser1 and testuser2 were created in the PDB database.

*Example 4-21 Displaying the usernames in the database*

```
SQL> sho pdbs

CON_ID CON_NAME                                OPEN MODE  RESTRICTED
-----
      3 DEVPDB                                READ WRITE NO
SQL> select username from dba_users where username in ('TESTUSER1','TESTUSER2');

USERNAME
-----
TESTUSER2
TESTUSER1
```

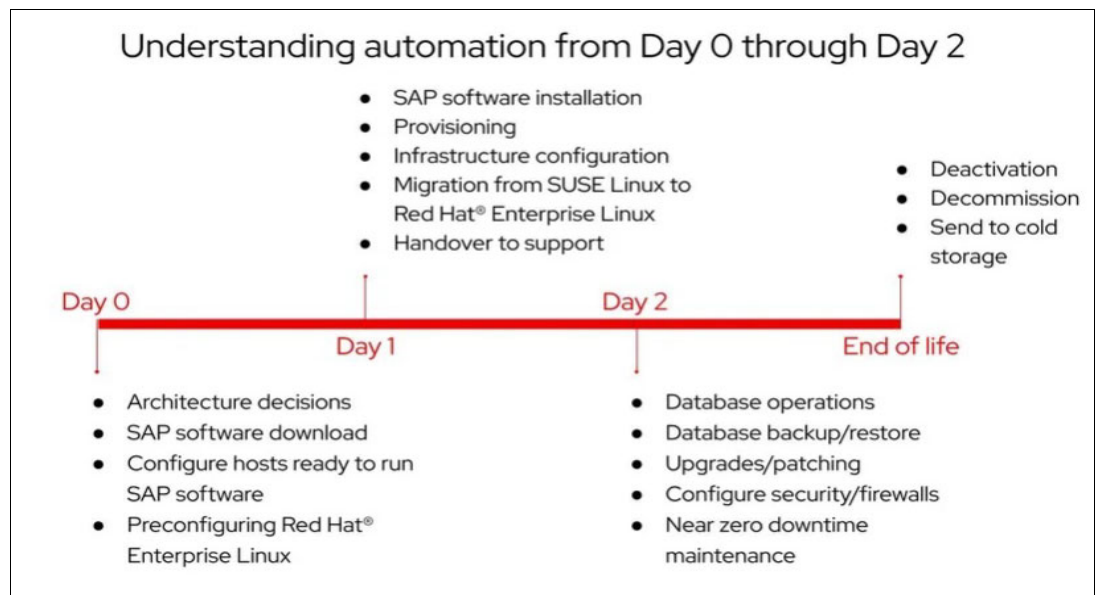
To run other playbooks, see the readme files at [GitHub](#) for each corresponding DB admin task.

## 4.7 SAP automation

If your organization relies on SAP HANA and S/4HANA for its critical business operations, downtime is not an option. However, the complexity of manually deploying and managing an SAP environment on-premises or in the cloud is time-consuming and error-prone, which can lead to service degradation, increased security exposure, and outages. Automation can help by performing the following functions:

- ▶ Streamline repetitive SAP management tasks.
- ▶ Help ensure consistent configuration across systems.
- ▶ Reduce deployment timelines and unplanned downtime.

Figure 4-9 describes how automation can help you over the lifecycle of your SAP environments.



*Figure 4-9 Automation benefits in your SAP environments<sup>5</sup>*

The end-to-end SAP S/4HANA installation process breaks down into four major blocks:

1. Server provisioning

This block is the most variable one because it depends on the infrastructure that is used. Server provisioning can be done by using Ansible alone or with other tools such as Terraform. Because SAP can be installed across many infrastructure and cloud environments, server provisioning is highly dependent on the infrastructure that is chosen for the SAP environment.

2. Basic OS setup

SAP has spent much effort in understanding how to install the base OS for the servers running the different SAP components. For SAP HANA, the OS is Linux, either SUSE or RHEL, and there are specific documented settings that are defined in multiple SAP notes. Likewise, there are documented settings for NetWeaver installations.

3. HANA installation and configuration.

4. S/4 installation and configuration.

Figure 4-10 illustrates the usage of Ansible for an SAP installation.

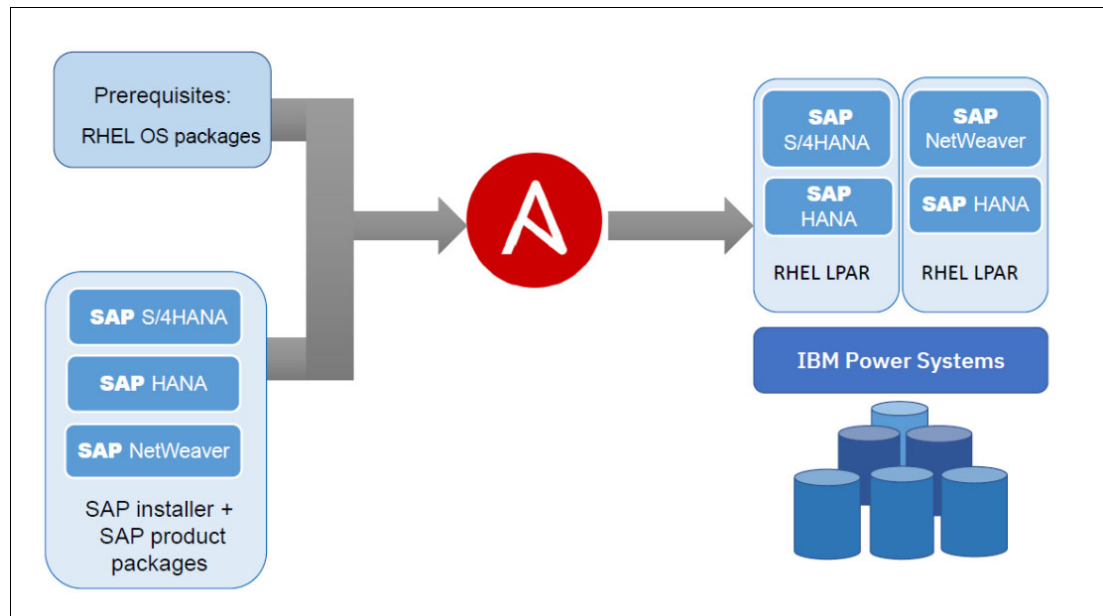


Figure 4-10 Ansible automation for an SAP installation<sup>6</sup>

<sup>5</sup> Source: <https://catalog.redhat.com/solutions/detail/e95cc4e4b41347639b8f5da129f588ac>

<sup>6</sup> Source: <https://community.sap.com/t5/technology-blogs-by-sap/automating-the-installation-of-sap-s-4hana-and-sap-hana-on-ibm-power/ba-p/13462363>

## SAP and Red Hat automation options

SAP and Red Hat have worked together for more than two decades to provide new innovative solutions for SAP users. For SAP automation with Ansible, there are two different options:

- ▶ Red Hat Enterprise Linux System Roles for SAP

[Red Hat Enterprise Linux System Roles for SAP](#) is a subset of the Red Hat Enterprise Linux System Roles, which is a collection of Ansible roles and modules that is provided as part of your Red Hat subscription. These system roles provide a stable and consistent configuration interface to automate and manage system functions across multiple releases of RHEL. These roles are run by Ansible to help administrators with server configuration after the servers are installed. The system roles are available either from the `rhel-system-roles` Red Hat Package Manager or from [Red Hat Automation Hub](#).

The roles are based on the development of the Linux System Roles upstream project, and for the SAP related roles, the SAP LinuxLab upstream project. These roles are supported by Red Hat.

- ▶ SAP LinuxLab Automation for SAP

The [SAP LinuxLab open-source initiative](#) provides simpler creation and management of SAP environments by using code and tools that are created by SAP Technology Partners. All capabilities that are available through the SAP LinuxLab open-source initiative are available for SAP Customers and SAP Service Partners. The projects within this initiative help SAP technical administrators and infrastructure administrators run their SAP systems by providing the following features:

- Automation of SAP Landscapes infrastructure, OS preparation, and SAP software installation with homogeneous architectures
- Automation of common operational tasks
- Sizing tools and architectural guidance for SAP Landscapes
- Other technical deliverables and tools

These projects are open source and community-supported and might have more up-to-date content compared to the Red Hat Enterprise Linux System Roles for SAP because it takes some time to integrate content from this GitHub project into the supported Red Hat product.

SAP LinuxLab has many projects and tools, but we focus in this publication on those projects that help automate tasks in the SAP environment, including installation or Day 0 operations, and Day 1 and Day 2 operations. The list of SAP LinuxLab projects that are relevant to automation is provided in Table 4-3.

Table 4-3 SAP LinuxLab projects

Project repository	Project description
<a href="#">community.sap_install</a>	Collection of Ansible Roles for various SAP software installations.
<a href="#">community.sap_operations</a>	Collection of Ansible Roles for various operational tasks with SAP Systems.
<a href="#">community.sap_launchpad</a>	Collection of Ansible Roles and Ansible Modules for various tasks by using SAP Launchpad APIs.
<a href="#">community.sles-for-sap</a>	Collection of Ansible roles for SUSE Linux Enterprise Server for SAP.
<a href="#">terraform.templates_for_sap</a>	Terraform Templates for deployment of various SAP solution scenarios for every cloud and hypervisor.



Project repository	Project description
<a href="#">terraform.modules_for_sap</a>	Terraform Modules for each cloud and hypervisor and dynamic Ansible Playbooks for SAP installations. Subcomponent of the Terraform Templates for SAP.
<a href="#">demo.sap_install</a>	Demonstration usage of the <code>community.sap_install</code> collection in Ansible Automation Platform or AWX.

Both of these options are described more completely in the following sections.

## 4.7.1 Red Hat Enterprise Linux System Roles for SAP

The Red Hat System Roles for SAP was introduced in RHEL 7. It is provided as part of the RHEL for SAP Solutions subscription, and can be used by Ansible to manage RHEL systems. With system roles, you can do the following tasks:

- ▶ Provision and configure infrastructure and network components, OS, and applications according to SAP HANA and SAP S/4HANA requirements.
- ▶ Manage configurations for RHEL, SAP, and any other system that integrates with your SAP deployment.
- ▶ Standardize infrastructure provisioning and configuration processes across your environment.
- ▶ Automate error-prone manual tasks to improve accuracy and reliability.
- ▶ Patch and perform other security-related operations.

The RHEL subscription supports RHEL System Roles. Here are the roles that are provided by the System Roles for SAP:

- ▶ `sap_general_preconfigure` (was named `sap-preconfigure` in earlier versions)
- ▶ `sap_netweaver_preconfigure` (was named `sap-netweaver-preconfigure` previously)
- ▶ `sap_hana_preconfigure` (was named `sap-hana-preconfigure` previously)

The RHEL System Roles for SAP, like the RHEL System Roles, are installed and run from a central node or control node. The control node connects to one or more managed nodes and performs installation and configuration steps on them. As a best practice, use the latest major release of RHEL on the control node (RHEL 8) and use the latest version of the roles either from the `rhel-system-roles-sap` Red Hat Package Manager or from Red Hat Automation Hub. The RHEL System Roles for SAP and Ansible packages do not need to be installed on the systems that are managed.

Table 4-4 shows the supported combinations of managed systems and control nodes for the current version of the Linux System Roles for SAP.

*Table 4-4 Supported configurations for System Roles for SAP*

Control node	Managed node	Support status
RHEL 8.4 or later	RHEL 8.0 or later	Fully supported
RHEL 8.4 or later	RHEL 7.6 or later	Fully supported
RHEL 8.4 or later	RHEL 7.5 or earlier	Not supported
RHEL 8.3 or earlier	RHEL (any release)	Not supported

**Note:** For control nodes running RHEL 7.8, RHEL 7.9, or RHEL 8.1, you can use the previous versions of `rhel-system-roles-sap`, which are in technology preview support status. For more information, see [Red Hat Enterprise Linux System Roles for SAP](#).

For control nodes running RHEL 8.2 or RHEL 8.3, you can use Version 2 of `rhel-system-roles-sap`, which is fully supported. For more information, see [Red Hat Enterprise Linux System Roles for SAP v2](#).

System Roles for SAP support multiple hardware architectures for the managed nodes, including `x86_64` for Intel compatible nodes, `ppc64le` for IBM Power nodes, and `s390x` for IBM Z servers.

**Important:** The System Roles for SAP are used right after the initial installation of a managed node. Do not run these roles against an SAP or other production system. The role enforces a certain configuration on the managed node, which might not be intended. Starting with Version 3, the roles support an `Assert` parameter for validating existing systems. For more information, see “Assert parameter” on page 227.

Before applying the roles on a managed node, verify that the RHEL release on the managed node is supported by the SAP software version that you are planning to install.

## Configuring SAP systems

The System Roles for SAP roles are designed to set up your SAP systems based on specific SAP documents. Table 4-5 defines what each of the roles are designed to do.

Table 4-5 Role descriptions and use cases

System role	Purpose	Use cases
<code>sap_general_preconfigure</code>	Install the software and perform all configuration steps that are required for the installation of SAP NetWeaver or SAP HANA.	SAP NetWeaver and SAP HANA
<code>sap_netweaver_preconfigure</code>	Install additional software and perform additional configuration steps that are required for SAP NetWeaver.	SAP NetWeaver
<code>sap_hana_preconfigure</code>	Install additional software and perform additional configuration steps that are required for SAP HANA.	SAP HANA

To prepare a managed node for running SAP HANA, run both the `sap_general_preconfigure` role and the `sap_hana_preconfigure` role. Likewise, to prepare a node to run SAP NetWeaver, run the `sap_general_preconfigure` role and the `sap_netweaver_preconfigure` role.

Table 4-6 shows the SAP Notes that are implemented by each of the system roles.

Table 4-6 Actions that are performed and SAP Notes that are implemented

System role	SAP Note for RHEL 7	SAP Note for RHEL 8
<code>sap_general_preconfigure</code>	SAP Note 2002167. SAP Note 1391070. SAP Note 0941735 (TMPFS only).	SAP Note 2772999.
<code>sap_netweaver_preconfigure</code>	SAP Note 2526952 (tuned profiles only).	SAP Note 2526952 (tuned profiles only).

System role	SAP Note for RHEL 7	SAP Note for RHEL 8
sap_hana_preconfigure	<p>Install required packages per documents SAP HANA 2.0 running on RHEL 7.x and SAP HANA SPS 12 running on RHEL 7.x, which are attached to SAP Note 2009879.</p> <p>ppc64le only: Install additional required packages per <a href="#">IBM Power Systems service and productivity tools</a>.</p> <p>Perform configuration steps per documents SAP HANA 2.0 running on RHEL 7.x and SAP HANA SPS 12 running on RHEL 7.x, which are attached to SAP Note 2009879.</p> <p>ppc64le only: SAP Note 2055470.</p> <p>SAP Note 2292690.</p> <p>SAP Note 2382421.</p>	<p>Install the required packages for SAP HANA as mentioned in SAP Note 2772999.</p> <p>ppc64le only: Install additional required packages per <a href="#">IBM Power Systems service and productivity tools</a>.</p> <p>ppc64le only: SAP Note 2055470.</p> <p>SAP Note 2777782.</p> <p>SAP Note 2382421.</p>

### Assert parameter

Starting with Version 3 of the package, `rhel-system-roles-sap` supports running the roles in assert mode. In assert mode, managed nodes are not modified, but instead they report the compliance of a node with the applicable SAP notes.

When running playbooks that use the assert mode on previous versions of the roles, the assert parameters are ignored, which can modify the managed node instead of checking them. Help ensure that Version 3 of the package is used. In addition, check that the playbooks you are using are calling the roles from the correct location, which is `/usr/share/ansible/roles` by default.

### Preparing a system for a SAP HANA installation

To prepare a system that is named `hana-p11` for SAP HANA installation, you can follow these steps:

1. Verify that there is no production software running on the managed node. Generally, the RHEL system roles for SAP are run right after RHEL installation, so there is no production software.
2. Validate that you can connect to the managed node by using SSH without a password:

```
# ssh hana-p11 uname -a
```

3. Create a yml file that is named `sap-hana.yml` that contains the content that is shown in Example 4-22.

*Example 4-22 Contents of sap-hana.yml*

---

```
- hosts: all
- vars:
  sap_preconfigure_reboot_ok: yes
  sap_hana_preconfigure_enable_sap_hana_repos: yes
  sap_hana_preconfigure_set_minor_release: yes
  sap_hana_preconfigure_modify_grub_cmdline_linux: yes
  sap_hana_preconfigure_reboot_ok: yes
roles:
  - sap_general_preconfigure
  - sap_hana_preconfigure
```

---

4. Run the following command:

```
# ansible-playbook -l hana-p11 sap-hana.yml
```

## 4.7.2 Using the SAP LinuxLab automation

There are multiple collections in the SAP LinuxLab open-source initiative. These collections can be combined to automate the full lifecycle of your SAP landscapes.

Most of the initial configuration and provisioning activity (Day 0 activities) is done with the Terraform modules and templates, which support many infrastructures, both on-premises and in the cloud. These options include support for on-premises Power servers and IBM Power Systems Virtual Server cloud instances.

### **community.sap\_install Ansible collection**

The Day 1 automated SAP installation activity is provided by the `community.sap_install` collection. This Ansible Collection runs various SAP Software installations and configuration tasks for running SAP software on Linux OSs. It includes handlers for SAP HANA Database Lifecycle Manager (HDBLCM) and SAP Software Provisioning Manager (SWPM), which enables programmatic deployment of any SAP solution scenarios. You can combine this collection with other Ansible Collections to provide end-to-end automation, from downloading SAP software installation media to the full configuration of SAP NetWeaver application servers and full high availability (HA) support by using built-in SAP HANA system replication technologies.

This Ansible Collection runs various SAP Software installations for different SAP solution scenarios, such as the following ones:

- ▶ SAP HANA installations through HDBLCM:
  - Install the SAP HANA database server with any SAP HANA Component (for example, Live Cache Apps, Application Function Library, or others).
  - Configure firewall rules and the hosts file for SAP HANA database server instances.
  - Apply for a license to SAP HANA.
  - Configure storage layouts for SAP HANA mount points (that is, `/hana/data`, `/hana/log`, and `/hana/shared`).
  - Install SAP Host Agent.
  - Install Linux Pacemaker, and configure Pacemaker Fencing Agents and Pacemaker Resource Agents.

- Install SAP HANA System Replication.
- Set high availability and disaster recovery (HADR) for SAP HANA System Replication.
- ▶ Every SAP Software installation through SAP SWPM:
  - Run software install tasks by using Ansible Variable to generate SWPM Unattended installations (use the `sap_swpm` Ansible Role default mode).
 

Optional usage of template definitions for repeated installations (use the `sap_swpm` Ansible Role default templates mode).
  - Run software install tasks with Ansible Variables one-to-one matched to SWPM Unattended `ini` file parameters to generate bespoke SWPM Unattended installations (use the `sap_swpm` Ansible Role advanced mode).
 

Optional usage of template definitions for repeated installations (use the `sap_swpm` Ansible Role advanced templates mode).
  - Run previously defined installations with an existing SWPM Unattended `inifile.params` (use the `sap_swpm` Ansible Role `inifile_reuse` mode).
  - Install Linux Pacemaker, and configure Pacemaker Fencing Agents and Pacemaker Resource Agents.
  - Set HADR with distributed SAP System installations (that is, SAP Replication Server (SRS))

The following SAP Software solutions were extensively tested:

- ▶ SAP HANA
  - Scale-Up
  - Scale-Out
  - HA
- ▶ SAP NetWeaver AS (Advanced Business Application Programming or Java) and other add-ons (for example, SAP GRC or SAP Adobe Document Services)
- ▶ SAP S/4HANA AnyPremise (1809, 1909, 2020, 2021, or 2022):
  - Sandbox (One Host) installation
  - Standard (Dual Host) installation
  - Distributed installation
  - HA installation
  - System Copy (Homogeneous with SAP HANA Backup / Recovery) installation
  - Maintenance Planner installation
  - System Rename
- ▶ SAP BW/4HANA
- ▶ SAP Business Suite on HANA (SAP Suite on HANA (SAP SoH), that is, SAP ECC on HANA)
- ▶ SAP Business Suite (that is, SAP ECC with SAP AnyDB - SAP ASE, SAP MaxDB, IBM Db2, or Oracle DB)
- ▶ SAP Solution Manager 7.2
- ▶ SAP Web Dispatcher

The collection is designed for Linux OSs. It has not been tested or adapted for SAP NetWeaver Application Server instances on IBM AIX or Windows Server. It supports RHEL 7 and later and SUSE Linux Enterprise Server 15 SP3 and later.

**Restriction:** The collection does not support SUSE Linux Enterprise Server before version 15 SP3 because of the following reasons:

- ▶ firewalld, which was added in SUSE Linux Enterprise Server 15 SP3, is used within the Ansible Collection.
- ▶ SELinux is used within the Ansible Collection. Full support for SELinux was provided as of SUSE Linux Enterprise Server 15 SP3.

This collection provides the Ansible roles that are described in Table 4-7. There are no custom modules.

*Table 4-7 Ansible roles that are provided by the installation collection*

Name	Summary
sap_anydb_install_oracle	Install Oracle DB 19.x for SAP.
sap_general_preconfigure	Configure general OS settings for SAP software.
sap_ha_install_hana_hsr	Install SAP HANA System Replication.
sap_ha_pacemaker_cluster	Install and configure Pacemaker and SAP resources.
sap_hana_install	Install SAP HANA through HDBLCM.
sap_hana_preconfigure	Configure settings for SAP HANA database server.
sap_hostagent	Install SAP Host Agent.
sap_hypervisor_node_preconfigure	Configure a hypervisor running VMs for SAP HANA.
sap_install_media_detect	Detect and extract SAP Software installation media.
sap_netweaver_preconfigure	Configure settings for SAP NetWeaver application server.
sap_storage_setup	Configure storage for SAP HANA with logical volume manager (LVM) partitions and XFS file system.
sap_swpm	Install SAP Software through SWPM.
sap_vm_preconfigure	Configure settings for a guest (VM) running on RHV/KVM for SAP HANA.

**Important:** In general, the “preconfigure” and “prepare” roles are prerequisites for the corresponding installation roles. The logic was separated to support a flexible execution of the different steps.

### The community.sap\_operations Ansible collection

This collection is designed to help with operational activity (Day 2) in your SAP landscape. Here are some of the use cases that can be automated with Ansible for Day 2 operations:

- ▶ SAP instance system copies.

- ▶ Spin up or delete new application servers on demand (for example, for hyperscalers).
- ▶ Instance refreshes.
- ▶ Kernel parameter changes.
- ▶ SAP kernel upgrade.
- ▶ DB operations.
- ▶ DB and OS patching.
- ▶ Resource addition (CPU, memory, and disk).
- ▶ Cluster management.
- ▶ DB backup/restore.
- ▶ Stop and start SAP instances.
- ▶ Shut down sandbox or preproduction systems to cold storage and pull them out of storage when needed.
- ▶ Smart management and proactive issue resolution for SAP servers.
- ▶ Near-zero downtime maintenance for SAP servers.

Day 2 operational automation is provided by the `community.sap_operations` collection. This Ansible Collection runs various SAP Systems operational tasks, which can be used individually during daily operations or combined for more complex automation of system maintenance activities. Here are some of the SAP Systems operational tasks:

- ▶ OS configuration postinstallation of SAP software:
  - Create an Ansible user for managing systems.
  - Update the `/etc/hosts` file.
  - Update the SSH authorized known hosts file.
  - Update the `fapolicy` entries based on SAP System instance numbers.
  - Update firewall port entries based on SAP System instance numbers.
  - License registration and refresh for RHEL subscription manager.
- ▶ SAP administration tasks
  - Start or stop of SAP HANA and SAP NetWeaver (in any configuration).
  - Update SAP profile files.
  - Run SAP Remote Function Calls (RFCs).

The collection consists of several Ansible Roles that combine several Ansible modules into a workflow. These roles, which are shown in Table 4-8, can be used within a playbook for specific tasks.

*Table 4-8 Ansible Roles*

Name	Summary
<code>os_ansible_user</code>	Creates an Ansible user <code>ansadm</code> with an SSH key.
<code>os_etchosts</code>	Updates <code>/etc/hosts</code> .
<code>os_knownhosts</code>	Updates the known hosts file <code>/.ssh/known_hosts</code> .
<code>sap_control</code>	Starts and stops SAP systems.
<code>sap_fapolicy</code>	Updates the service <code>fapolicyd</code> for generic, <code>sap nw</code> , or <code>sap hana</code> related UIDs.
<code>sap_firewall</code>	Updates the service <code>firewalld</code> for generic, <code>sap nw</code> , or <code>sap hana</code> related ports.

Name	Summary
sap_profile_update	Updates the default and instance profiles.
sap_rfc	Runs SAP RFCs.
sap_rhsm	Red Hat subscription manager registration.

In addition to the roles, there are more Ansible modules that are provided by the collection. These modules, which are shown in Table 4-9, can be called directly within a playbook.

Table 4-9 Ansible modules

Name	Summary
sap_operations.sap_facts	Gathers SAP facts in a host (for example, SAP System IDs and SAP System Instance Numbers of either SAP HANA database server or SAP NetWeaver application server).
sap_operations.sap_monitor_hana_status	Checks the status of a running SAP HANA database server.
sap_operations.sap_monitor_nw_status	Checks the status of a running SAP NetWeaver application server.
sap_operations.sap_monitor_nw_perf	Checks host performance metrics from an SAP NetWeaver Primary Application Server (PAS) instance.
sap_operations.sap_monitor_nw_response	Checks system response time metrics from an SAP NetWeaver PAS instance.

More download automation software is provided by the `community.sap_launchpad` collection.

### Example scenarios

This section provides some example scenarios that use the functions that are provided by the `community.sap_operations` Ansible collection.

#### ► sap\_control

This Ansible Role runs basic SAP administration tasks on Linux OSs:

- Start, stop, or restart SAP HANA Database Server.
- Start, stop, or restart SAP NetWeaver Application Server.
- Multiple automatic discoveries, and start, stop, or restart SAP HANA Database Server or SAP NetWeaver Application Server.

The specific control function is defined by using the `sap_control_function` parameter, which can be any of the following items:

- `restart_all_sap`
- `restart_all_nw`
- `restart_all_hana`
- `restart_sap_nw`
- `restart_sap_hana`
- `stop_all_sap`
- `start_all_sap`
- `stop_all_nw`
- `start_all_nw`
- `stop_all_hana`
- `start_all_hana`



- stop\_sap\_nw
- start\_sap\_nw
- stop\_sap\_hana
- start\_sap\_hana

Executions specifying all automatically detect any System IDs and corresponding Instance Numbers. To specify a specific SAP system, provide the SAP system SID as a parameter.

To restart all SAP systems, use the following parameter:

```
sap_control_function: "restart_all_sap"
```

To stop a specific SAP HANA database, use the following parameters:

```
sap_control_function: "stop_sap_hana"
sap_sid: "HDB"
```

► sap\_hana\_sr\_takeover

This role can be used to help ensure, control, and change SAP HANA System Replication. The role assumes that the SAP HANA System Replication was configured by using the community.sap\_install.sap\_ha\_install\_hana\_hsr role.

The variables that are shown in Table 4-10 are mandatory for running this role unless a default value is specified.

Table 4-10 Required variables

Variable name	Description
sap_hana_sr_takeover_primary	The server that becomes the primary server.
sap_hana_sr_takeover_secondary	The server that registers as a secondary server. The role can be run twice if more than one secondary server is needed by looping this variable.
sap_hana_sr_takeover_sitename	The name of the site that is registered as a secondary site.
sap_hana_sr_takeover_rep_mode	The HANA replication mode (defaults to sync if not set).
sap_hana_sr_takeover_hsr_oper_mode	The HANA replication operation mode (defaults to logreplay).
sap_hana_sid	The HANA SID.
sap_hana_instance_number	The HANA instance number.

The playbook that is shown in Example 4-23 shows how to implement this role. The assumption is that there are two systems (hana1 and hana2) that are set up for SAP HSR, with SID RHE and instance 00. The playbook helps ensure that hana1 is the primary and hana2 is the secondary. The role does nothing if hana1 is already the primary and hana2 is the secondary. The role fails if hana1 is not configured for system replication and not in sync.

Example 4-23 Playbook to set the primary in a HANA HSR setup

```
---
- name: Helps ensure hana1 is primary
  hosts: hanas
  become: true
  tasks:
    - name: Switch to hana1
```

```

ansible.builtin.include_role:
  name: community.sap_operations.sap_hana_sr_takeover
vars:
  sap_hana_sr_takeover_primary: hana2
  sap_hana_sr_takeover_secondary: hana1
  sap_hana_sr_takeover_sitename: DC01
  sap_hana_sid: "RHE"
  sap_hana_instance_number: "00"

```

---

More documentation and examples are available as part of the collection documentation at [GitHub](#). In the roles directory, in the GitHub location, there is a subdirectory for each role that includes a readme file that provides the specifics about how that role operates and its requirements.

### The community.sap\_launchpad Ansible collection

This Ansible Collection runs basic SAP.com Support operations tasks to help download SAP software or download files from the SAP Maintenance Planner. Maintenance Planner is a solution that is hosted by SAP that helps plan and maintain systems in an SAP landscape, and it can be used to plan complex activities like installing a new system or updating existing systems.

Here are some of the operations that are supported:

- ▶ Software Center Catalog:
  - Search and download SAP software center catalog files.
  - Search and extract SAP software center catalog information.
- ▶ Maintenance Planner: Look up and download files from an existing 'New Implementation' MP Transaction and Stack by using SAP the software center's download basket.

The collection provides the modules that are shown in Table 4-11.

Table 4-11 Ansible modules from community.sap\_launchpad

Name	Functions
sap_launchpad.software_center_download	Search for files and downloads.
sap_launchpad.maintenance_planner_files	Maintenance Planner files retrieval.
sap_launchpad.maintenance_planner_stack_xml_download	Maintenance Planner stack XML download.

**Note:** SAP software installation media must be obtained from SAP directly, which requires valid license agreements with SAP to access these files.

#### **Credentials: SAP User ID**

An SAP Company Number (SCN) contains one or more Installation Numbers, which provide licenses for specified SAP Software. When an SAP User ID is created within the SCN, the administrator must provide SAP Download authorizations for the SAP User ID.

When an SAP User ID is enabled as part of an SAP Universal ID, then the sap\_launchpad Ansible collection must use the following items:

- ▶ The SAP User ID
- ▶ The password for login with the SAP Universal ID

If an SAP Universal ID is used, then check and reset the SAP User ID 'Account Password' in the SAP Universal ID Account Manager to avoid any potential conflicts.

Example 4-24 provides an example playbook to download specific SAP software by using the `sap_launchpad.software_center-download` module. In this playbook, the user is prompted to enter the SAP user ID and password. The playbook may be modified to use variables to enter these values.

*Example 4-24 Sample playbook to download software from the SAP software center*

---

```
---
- hosts: all

collections:
  - community.sap_launchpad

pre_tasks:
  - name: Install Python package manager pip3 to system Python
    yum:
      name: python3-pip
      state: present
  - name: Install Python dependencies for Ansible Modules to system Python
    pip:
      name:
        - urllib3
        - requests
        - beautifulsoup4
        - lxml

# Prompt for Ansible Variables
vars_prompt:
  - name: suser_id
    prompt: Enter S-User
    private: no
  - name: suser_password
    prompt: Enter Password
    private: yes

# Define Ansible Variables
vars:
  ansible_python_interpreter: python3
  softwarecenter_search_list:
    - 'SAPCAR_1324-80000936.EXE'
    - 'HCMT_057_0-80003261.SAR'

# Use task block to call Ansible Module
tasks:
  - name: Run Ansible Module to download SAP software
    community.sap_launchpad.software_center_download:
      suser_id: "{{ suser_id }}"
      suser_password: "{{ suser_password }}"
      softwarecenter_search_query: "{{ item }}"
    dest: "/tmp/"
    loop: "{{ softwarecenter_search_list }}"
    loop_control:
      label: "{{ item }} : {{ download_task.msg }}"
```

```
register: download_task
retries: 1
until: download_task is not failed
```

---

Example 4-25 shows an example playbook that downloads a list of files that are defined by using the Maintenance Planner. The playbook prompts for your SAP user credentials and a specific Maintenance Planner transaction name that was previously created.

*Example 4-25 Playbook to download files that are defined in a Maintenance Planner transaction*

---

```
---
- hosts: all

collections:
  - community.sap_launchpad

# pre_tasks:

# Prompt for Ansible Variables
vars_prompt:
  - name: suser_id
    prompt: Enter S-User
    private: no
  - name: suser_password
    prompt: Enter Password
    private: yes
  - name: mp_transaction_name
    prompt: Enter MP transaction name
    private: no

# Define Ansible Variables
vars:
  ansible_python_interpreter: python3

# Use task block to call Ansible Module
tasks:
  - name: Run Ansible Module 'maintenance_planner_files' to get files from MP
    community.sap_launchpad.maintenance_planner_files:
      suser_id: "{{ suser_id }}"
      suser_password: "{{ suser_password }}"
      transaction_name: "{{ mp_transaction_name }}"
      register: sap_maintenance_planner_basket_register

  # - debug:
  #   msg:
  #     - "{{ sap_maintenance_planner_basket_register.download_basket }}"

  - name: Run Ansible Module 'software_center_download' to download files
    community.sap_launchpad.software_center_download:
      suser_id: "{{ suser_id }}"
      suser_password: "{{ suser_password }}"
      download_link: "{{ item.DirectLink }}"
      download_filename: "{{ item.Filename }}"
      dest: "/tmp/test"
    loop: "{{ sap_maintenance_planner_basket_register.download_basket }}"
    loop_control:
```

```
label: "{{ item }}" : {{ download_task.msg }}"  
register: download_task  
retries: 1  
until: download_task is not failed
```

---





# Infrastructure as Code by using Ansible

Infrastructure as Code (IaC) is the ability to automate the management of infrastructure resources by using code, for example, Ansible. Traditionally, an Ansible client is as a virtual machine (VM) or a storage controller, but with IaC, the Ansible client can be a service that manages end-to-end infrastructure resources, which include VMs, networks, storage, zoning, and other resources. Within IBM Power environments, these services include IBM PowerVC, IBM Cloud Power Systems Virtual Server, and IBM PowerVM, which provide Virtual I/O Server (VIOS) management and the Hardware Management Console (HMC).

You can use IaC to deploy, destroy, resize, rebalance, and migrate workloads to different infrastructures. This chapter introduces the concept of IaC and describes the capabilities that Ansible provides to deliver IaC on IBM Power.

The following topics are described in this chapter:

- ▶ IBM Power Virtualization Center
- ▶ IBM Power Systems Virtual Server

## 5.1 IBM Power Virtualization Center

IBM Power Virtualization Center (PowerVC) provides simplified management of IBM AIX, IBM i, and Linux VMs running on IBM Power. It is built on OpenStack to provide private cloud capabilities across your IBM Power environment. IBM PowerVC capabilities include creating and destroying VMs, networks, network interfaces, storage volumes, and images. It can also perform tasks against the VMs, such as stop, start, resize, migrate, clone, create, and restore snapshots, and attach storage volumes.

### 5.1.1 Advantages of PowerVC

PowerVC offers a range of benefits that are tailored to IBM Power environments:

- ▶ **Expandability:** Attach volumes or more networks to VMs.
- ▶ **Flexibility:** Import and export existing systems and volumes between on-premises and off-premises locations.
- ▶ **Efficiency:** Take snapshots of VMs and clone them for quick replication.
- ▶ **Seamless migration:** Migrate running VMs by using Live Partition Mobility (LPM).
- ▶ **Continuity:** Restart VMs remotely if there is a server failure.
- ▶ **Simplified management:** Streamline Power virtualization administration.
- ▶ **Agility:** Adapt swiftly to changing business requirements.
- ▶ **Dynamic resource management:** Create, resize, and adjust CPU and memory resources for VMs.

When you deploy a new VM by using IBM PowerVC, it performs all the required tasks:

- ▶ Creates the VM profile on the HMC, including all network and storage interfaces.
- ▶ Creates the appropriate storage area network (SAN) zoning.
- ▶ Creates the VM on the storage controller.
- ▶ Creates the root and non-root storage volumes.
- ▶ Updates the VIOS to map the VM to its new volumes.
- ▶ Starts the new VM.

When you delete a VM, all the resources that were created by IBM PowerVC are removed cleanly.

With IBM PowerVC, there are two options to choose from to work with Ansible:

- ▶ Using the OpenStack modules
- ▶ Making Representational State Transfer (REST) application programming interface (API) calls by using the URI module

This section covers both methods.

### 5.1.2 Using the OpenStack Cloud modules

In this section, you learn about the following IaC options that use the OpenStack Cloud modules and IBM PowerVC:

- ▶ Authentication
- ▶ Creating a VM
- ▶ Destroying an existing VM
- ▶ Showing resource information



- ▶ Stopping or starting a VM
- ▶ Creating a storage volume
- ▶ Attaching a storage volume to an existing VM

Because IBM PowerVC is built on OpenStack, you may use several of the cloud modules that are provided by the OpenStack community. These modules are available at [Ansible Galaxy](#).

You can download the OpenStack Cloud collection either by using `ansible-galaxy` from the CLI or by using the `requirements.yml` file. Example 5-1 shows using `ansible-galaxy` to download the collection.

*Example 5-1 Downloading the OpenStack Cloud collection*

---

```
ansible-galaxy collection install openstack.cloud
Process install dependency map
Starting collection install process
Installing 'openstack.cloud:2.1.0' to
'/root/.ansible/collections/ansible_collections/openstack/cloud'
```

---

Example 5-2 shows the `requirements.yml` file that you can use to download the collection.

*Example 5-2 Example requirements.yml file to download the OpenStack Cloud collection*

---

```
collections:
  - name: openstack.cloud
    source: https://galaxy.ansible.com
```

---

For Ansible to run the OpenStack Cloud modules, first install the OpenStack SDK on your Ansible Controller, as described in the ‘readme’ section of the collection page at Ansible Galaxy. The command that is shown in Example 5-3 installs the SDK.

*Example 5-3 Downloading the OpenStack Cloud collection*

---

```
$ pip3.9 install openstacksdk
```

---

You can verify that the modules were installed correctly by using the `ansible-doc` command. Example 5-4 shows an example of the documentation for the OpenStack Cloud image information module.

*Example 5-4 Viewing the openstack.cloud.server\_info documentation*

---

```
$ ansible-doc openstack.cloud.image_info
..
REQUIREMENTS: python >= 3.6, openstacksdk >= 1.0.0
AUTHOR: OpenStack Ansible SIG
EXAMPLES:
- name: Gather previously created image named image1
  openstack.cloud.image_info:
    cloud: devstack-admin
    image: image1

- name: List all images
  openstack.cloud.image_info:
```

---

Table 5-1 shows some of the OpenStack Cloud modules that are relevant to IBM PowerVC.

Table 5-1 OpenStack Cloud modules

Modules name	Function
openstack.cloud.auth	Retrieves an auth token from PowerVC Cloud.
openstack.cloud.compute_flavor	Manages PowerVC compute flavors.
openstack.cloud.compute_flavor_info	Fetches compute flavors information from PowerVC Cloud.
openstack.cloud.image	Manages PowerVC images.
openstack.cloud.image_info	Fetches image information from PowerVC Cloud.
openstack.cloud.keypair	Manages PowerVC keypairs.
openstack.cloud.keypair_info	Fetches keypair information from PowerVC Cloud.
openstack.cloud.project	Manages PowerVC projects.
openstack.cloud.project_info	Fetches project information from PowerVC Cloud.
openstack.cloud.server	Creates or deletes VMs within PowerVC Cloud.
openstack.cloud.server_action	Performs actions on PowerVC VMs.
openstack.cloud.server_info	Fetches VM information from PowerVC Cloud.
openstack.cloud.server_volume	Attaches or detaches volumes from PowerVC VMs.
openstack.cloud.volume	Manages PowerVC storage volumes.
openstack.cloud.volume_info	Fetches storage volume information from PowerVC Cloud.
openstack.cloud.volume_snapshot	Manages PowerVC snapshots.
openstack.cloud.volume_snapshot_info	Fetches snap host information from PowerVC Cloud.

## Authenticating with IBM PowerVC

Before Ansible can use the OpenStack cloud modules, it must authenticate with the IBM PowerVC server.

To authenticate from the CLI, create a `clouds.yaml` file that contains the information about the cloud environments that Ansible must connect to. In this case, the environment is an IBM PowerVC server. The OpenStack modules look for the `clouds.yaml` file in the following directories:

- ▶ current directory
- ▶ `~/.config/openstack`
- ▶ `/etc/openstack`

The modules use the first one it finds.

The contents of an example `clouds.yaml` file are shown in Example 5-5 on page 243.

#### Example 5-5 Example clouds.yml file

---

```
$ cat clouds.yml
powervc_cloud:
  auth:
    auth_url: https://x.x.x.x:5000/v3/
    project_name: ibm-default
    project_domain_name: Default
    user_domain_name: Default
    username: <powervc_userid>
    password: <powervc_userid_password>
    region_name: RegionOne
    cacert: "./powervc.crt"
```

---

The first line is the name of your cloud. The name is for reference only, and it does not have to match the real name. You can use the cloud name to define authentication methods to multiple OpenStack clouds and refer to them individually within your playbooks.

The `auth_url` is the IP address of your IBM PowerVC server, and the remaining auth settings are specific to your PowerVC environment, such as project, user ID, password, and others.

You must have a copy of the CA cert file from the PowerVC server that you reference in the `cacert` line, which can be found on your PowerVC server (the default location is `/etc/pki/tls/certs/powervc.crt`).

### Confirming authentication to IBM PowerVC

Now that you have set up authentication, you can confirm that Ansible can use the OpenStack cloud modules to retrieve information from IBM PowerVC. A simple way to do this task is to create a playbook to show the images that are contained on IBM PowerVC, as shown in Example 5-6.

#### Example 5-6 Example playbook to list PowerVC images by using the `openstack.cloud.image_info` module

---

```
$ cat PowerVC_list_images.yml
---
- name: List available PowerVC Images
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Retrieve list of all AIX images
      openstack.cloud.image_info:
        cloud: powervc_cloud
        register: image_results

    - name: Show name, ID, OS distribution, and status of images
      debug:
        msg: "{{ image_results | json_query('images[*].
              {name: name, id: id, os_distro: os_distro, status: status}') }}"
```

---

In Example 5-7, you call a task by using the `openstack.cloud.image_info` modules to point at the cloud `powervc_cloud`. This cloud name must match the entry in your `clouds.yml` authentication file that you defined earlier. Then, register the results and output them in the second task. Now, you can run the playbook to list your IBM PowerVC images, as shown in Example 5-7.

*Example 5-7 An example playbook to list PowerVC images by using the `openstack.cloud.image_info` module*

```
$ ansible-playbook PowerVC_list_images.yml
PLAY [List available PowerVC Images]
*****
TASK [Retrieve list of all AIX images]
*****
ok: [localhost]
TASK [Show name, ID, OS distribution, and status of images]
*****
ok: [localhost] => {
  "msg": [
    {
      "id": "f62a76dd-4742-445f-aa5c-f3f447dd778e",
      "name": "RHCOS-4.12.17",
      "os_distro": "coreos",
      "status": "active"
    },
    {
      "id": "0930d057-dc7e-415f-97cd-1fe36ecdcbd",
      "name": "RHEL v9.1",
      "os_distro": "rhel",
      "status": "active"
    },
    {
      "id": "d51d8cfd-c83b-4ec6-9464-8d4215259546",
      "name": "AIX 7.3",
      "os_distro": "aix",
      "status": "active"
    },
    {
      "id": "c64ff508-3a81-4679-a9e4-29acc3f96430",
      "name": "IBM i v7.3",
      "os_distro": "ibmi",
      "status": "active"
    }
  ]
}

PLAY RECAP
*****
localhost : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

**Note:** You might have to install the `community.general` collection to parse the data by using `json_query`, as shown in Example 5-8.

*Example 5-8 Downloading the Community General collection by using ansible-galaxy*

---

```
$ ansible-galaxy collection install community.general
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/community-general-7.2.1.tar.gz to
/root/.ansible/tmp/ansible-local-3467397n4zdigf9/tmpc_m7f9qt/community-general-7.2
.1-i83jeq8b
Installing 'community.general:7.2.1' to
'/root/.ansible/collections/ansible_collections/community/general'
community.general:7.2.1 was installed successfully
```

---

## Creating VMs by using the openstack.cloud.server module

You can use the `openstack.cloud.server` module to create an AIX, IBM i, or Linux VM, as shown in Example 5-9.

*Example 5-9 Creating a VM by using PowerVC*

---

```
- name: Creating VM {{ VM_Name }} by using PowerVC
  openstack.cloud.server:
    cloud: "{{ PowerVC_Cloud_Name }}"
    state: present
    name: "{{ VM_Name }}"
    image: "{{ Image_ID_or_Name }}"
    flavor: "{{ Flavor_ID_or_Name }}"
    network: "{{ Network_Name }}"
    key_name: "{{ SSH-Key_Name }}"
  register: vm_create_information
```

---

In Example 5-9, you passed the `openstack.cloud.server` module a few key variables so that it could build the VM:

<cloud>	The name of the PowerVC cloud that is defined in <code>clouds.yml</code> .
<state>	Present (if the VM does not exist, create it).
<name>	The name of the new VM (in this example, <code>aix-vm-1</code> ).
<image>	The name or ID of the PowerVC image to use (obtained from PowerVC).
<flavor>	The name or IR of the PowerVC compute flavor to use (obtained from PowerVC)
<network>	The name of the PowerVC network to use (obtained from PowerVC).
<key_name>	The name of the Secure Shell (SSH) key pair to inject into the new VM (obtained from PowerVC).

When you run this task, Ansible uses the `openstack.cloud.server` module to connect to the IBM PowerVC cloud and create the VM by using the name that is provided. The results for running the playbook are shown in Example 5-10.

*Example 5-10 Output from creating a VM by using OpenStack modules on IBM PowerVC*

```
PLAY [Connect to PowerVC/Openstack and build VM]
*****

TASK [Creating VM aix-vm-1 by using PowerVC]
*****
changed: [localhost]
PLAY RECAP *****
localhost : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

You can also see the new VM being created on the IBM PowerVC UI, as shown in Figure 5-1.

**Note:** In Example 5-10, you allowed PowerVC to assign an IP address from its IP pool.

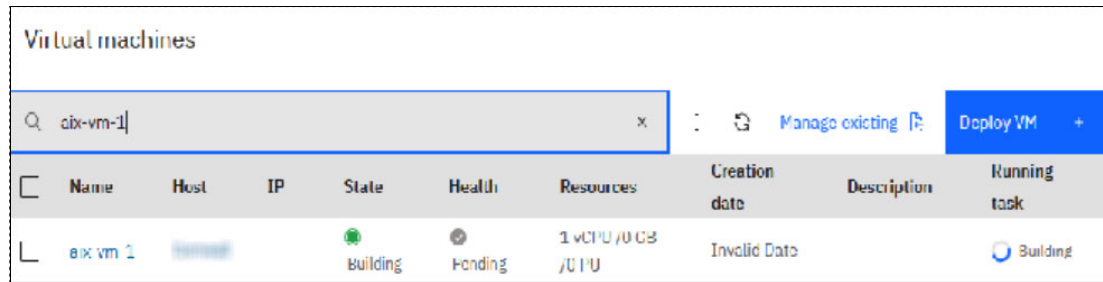


Figure 5-1 PowerVC VM build by using OpenStack modules

If any of the variables that are passed are incorrect, the module fails without creating the VM. For example, if you passed the module an incorrect image name, it fails with a message similar to what is shown in Example 5-11.

*Example 5-11 Error showing an incorrect PowerVC image name*

```
TASK [Create a VM instance by using PowerVC]
*****
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Could not find image AIX 7.2 TL2 SP6"}
```

**Destroying a VM by using the `openstack.cloud.server` module**

You can also use the `openstack.cloud.server` module to destroy an AIX, IBM i, or Linux VM, as shown Example 5-12.

*Example 5-12 Destroying an existing VM by using PowerVC*

```
- name: "Destroying VM {{ VM_Name }} by using PowerVC"
  openstack.cloud.server:
    cloud: "{{ PowerVC_Cloud_Name }}"
    state: absent
    name:   "{{ VM_Name }}"
    register: vm_destroy_information
```

In Example 5-12 on page 246, you passed the `openstack.cloud.server` module three variables to destroy the VM:

```
<cloud>           The name of the PowerVC cloud that is defined in clouds.yml.
<state>          Absent (if the VM exists, remove it).
<name>           The name of the existing VM to destroy (in this example, we use
                  aix-vm-1).
```

Because the VM is managed by IBM PowerVC, when it is destroyed, all its resources, including the storage volumes, the SAN zones, and its IP address allocations, are also removed by default.

When you use the `openstack.cloud.server` module to destroy an existing VM, you receive the message that the status changed, as shown in Example 5-13.

*Example 5-13 Message showing that an existing VM was destroyed by using PowerVC*

```
TASK [Destroying VM aix-vm-1 by using PowerVC]
*****
changed: [localhost]

TASK [Show VM destroy output]
*****
ok: [localhost] => {
  "vm_destroy_information": {
    "changed": true,
    "failed": false
  }
}
PLAY RECAP *****
localhost : ok=3   changed=1   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0
```

You can also see that the new VM was destroyed on the IBM PowerVC UI, as shown in Figure 5-2.

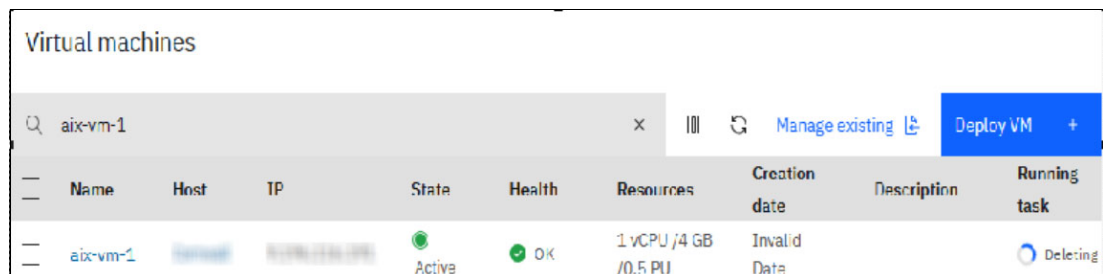


Figure 5-2 PowerVC UI showing that the VM was destroyed by using OpenStack modules

If you try to destroy a VM that does not exist, the `openstack.cloud.server` modules do not return an error because you stated that the VM should be 'state: absent'. The output tells you that the status did not change (false), as shown in Example 5-14.

*Example 5-14 Message showing an attempt to destroy a VM that does not exist by using PowerVC*

```
TASK [Destroying VM aix-vm-1 by using PowerVC]
*****
ok: [localhost]

TASK [Show VM destroy output]
*****
ok: [localhost] => {
```

```

    "vm_information": {
        "changed": false,
        "failed": false
    }
}

```

---

## Retrieving IBM PowerVC resource information by using the openstack.cloud modules

You can also use the `openstack.cloud` module to retrieve data from your PowerVC cloud. You can see some of the common 'information' modules in Table 5-1 on page 242, including `openstack.cloud.server_info`, `openstack.cloud.volume_info`, and `openstack.cloud.image_info`. Because the `openstack.cloud` modules retrieve their data from the PowerVC server and not the VM, you can query the resources infrastructure.

### Displaying all VM information by using PowerVC

You can use `openstack.cloud.server_info` to retrieve infrastructure information about a PowerVC controlled VM. Example 5-15 shows the module collecting all the information that PowerVC knows about a VM.

#### Example 5-15 Displaying all the VM information

```

- name: Retrieve all information about an existing VM instance by using PowerVC
  openstack.cloud.server_info:
    cloud: "{{ PowerVC_Cloud_Name }}"
    name:   "{{ VM_Name }}"
    register: vm_information

- name: Show all VM information collected
  var: vm_information

```

---

The output that is collected shows a large amount of information that PowerVC retrieved about the VM, which is shown in Example 5-16.

#### Example 5-16 Output from openstack.cloud.server\_info

Output of information retrieval about a VM by using `openstack.cloud.server_info`

```

"servers": [
  {
    "access_ipv4": "x.x.x.x",
    "access_ipv6": "",
    "addresses": {
      "VLAN_888-NET_116": [
        {
          "OS-EXT-IPS-MAC:mac_addr": "fa:5f:75:e3:yy:xx",
          "OS-EXT-IPS:type": "patched",
          "addr": "x.x.x.x",
          "version": 4
        }
      ]
    },
    "admin_password": null,
    "attached_volumes": [
      {
        "attachment_id": null,
        "bdm_id": null,

```



```

        "delete_on_termination": true,
        "device": null,
        "id": "5119dfc1-8fc2-4a70-943d-da6266d71f9b",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
    },
    {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": false,
        "device": null,
        "id": "25d70a6e-0923-491c-bb87-47b484b11c16",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
    }
],
"availability_zone": "Default Group",
"block_device_mapping": null,
"compute_host": "828422A_XXXXXX",
"config_drive": "",
"created_at": "2023-06-28T09:09:50Z",
"description": "aix-vm-1",
"disk_config": "MANUAL",
"fault": null,
"flavor": {
    "description": null,
    "disk": 0,
    "ephemeral": 0,
    "extra_specs": {
        "powervm:availability_priority": "127",
        "powervm:dedicated_proc": "false",
        "powervm:enable_lpar_metric": "true",
        "powervm:enforce_affinity_check": "false",
        "powervm:max_mem": "4096",
        "powervm:max_proc_units": "0.5",
        "powervm:max_vcpu": "1",
        "powervm:min_mem": "2048",
        "powervm:min_proc_units": "0.1",
        "powervm:min_vcpu": "1",
        "powervm:proc_units": "0.1",
        "powervm:processor_compatibility": "default",
        "powervm:secure_boot": "0",
        "powervm:shared_proc_pool_name": "DefaultPool",
        "powervm:shared_weight": "128",
        "powervm:srr_capability": "true",
        "powervm:uncapped": "true"
    },
    "id": "xtiny",
    "is_disabled": null,
    "is_public": true,
    "location": null,

```

```

        "name": "xtiny",
        "original_name": "xtiny",
        "ram": 4096,
        "rxtx_factor": null,
        "swap": 0,
        "vcpus": 1
    },
    "flavor_id": null,
    "has_config_drive": "",
    "host_id":
"6e82dcb4ed92b0e70c305e2ee1021f0019d3bd88e9dd910b5a81xxxx",
    "host_status": "UP",
    "hostname": "aix-vm-1",
    "hypervisor_hostname": "XXXXX",
    "id": "371aa5fe-b5c2-4660-978b-09b323a49f66",
    "image": {
        "architecture": null,
        "checksum": null,
        "container_format": null,
        "created_at": null,
        "direct_url": null,
        "disk_format": null,
        "file": null,
        "has_auto_disk_config": null,
        "hash_algo": null,
        "hash_value": null,
        "hw_cpu_cores": null,
        "hw_cpu_policy": null,
        "hw_cpu_sockets": null,
        "hw_cpu_thread_policy": null,
        "hw_cpu_threads": null,
        "hw_disk_bus": null,
        "hw_machine_type": null,
        "hw_qemu_guest_agent": null,
        "hw_rng_model": null,
        "hw_scsi_model": null,
        "hw_serial_port_count": null,
        "hw_video_model": null,
        "hw_video_ram": null,
        "hw_vif_model": null,
        "hw_watchdog_action": null,
        "hypervisor_type": null,
        "id": "71c5ddb5-f4f9-431b-917d-e0c0df581xxx",
        "instance_type_rxtx_factor": null,
        "instance_uuid": null,
        "is_hidden": null,
        "is_hw_boot_menu_enabled": null,
        "is_hw_vif_multiqueue_enabled": null,
        "is_protected": null,
        "kernel_id": null,
        "location": null,
        "locations": null,
        "metadata": null,
        "min_disk": null,
        "min_ram": null,

```

```

    "name": null,
    "needs_config_drive": null,
    "needs_secure_boot": null,
    "os_admin_user": null,
    "os_command_line": null,
    "os_distro": null,
    "os_require_quiesce": null,
    "os_shutdown_timeout": null,
    "os_type": null,
    "os_version": null,
    "owner": null,
    "owner_id": null,
    "properties": {
      "links": [
        {
          "href":
"https://x.x.x.x:8774/6a01a6c6f13c40f79b7ff55xxxx70a371/images/71c5ddb5-f4f9-431b-
917d-e0c0xxx",
          "rel": "bookmark"
        }
      ]
    },
    "ramdisk_id": null,
    "schema": null,
    "size": null,
    "status": null,
    "store": null,
    "tags": [],
    "updated_at": null,
    "url": null,
    "virtual_size": null,
    "visibility": null,
    "vm_mode": null,
    "vmware_adaptertype": null,
    "vmware_ostype": null
  },
  "image_id": null,
  "instance_name": "aix-vm-1-371aa5fe-00000b9e",
  "is_locked": false,
  "kernel_id": "",
  "key_name": "ssh-key",
  "launch_index": 0,
  "launched_at": "2023-06-28T09:13:22.000000",
},
"max_count": null,
"metadata": {
  "enforce_affinity_check": "false",
  "hostname": "aix-vm-1",
  "move_pin_vm": "false",
  "original_host": "828422A_xxxx",
},
"min_count": null,
"name": "aix-vm-1",
"networks": null,
"power_state": 1,

```

```

    "progress": 100,
    "project_id": "6a01a6c6f13c40f79b7ff5552170axxx",
    "ramdisk_id": "",
    "reservation_id": "r-4n2mezi3",
    "root_device_name": "/dev/sda",
    "scheduler_hints": null,
    "security_groups": null,
    "server_groups": null,
    "status": "ACTIVE",
    "tags": [],
    "task_state": null,
    "terminated_at": null,
    "trusted_image_certificates": null,
    "updated_at": "2023-08-15T13:08:46Z",
    "user_data": null,
    "user_id":
"0688b01e6439ca32d698d20789d52169126fb41fb1a4ddafcebb97d854e836c9",
    "vm_state": "active",
    "volumes": [
      {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": true,
        "device": null,
        "id": "5119dfc1-8fc2-4a70-943d-da6266d71f9b",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
      },
      {
        "attachment_id": null,
        "bdm_id": null,
        "delete_on_termination": false,
        "device": null,
        "id": "25d70a6e-0923-491c-bb87-47b484b11c16",
        "location": null,
        "name": null,
        "tag": null,
        "volume_id": null
      }
    ]

```

---

### ***Displaying only VMs that are hosted on a specific IBM Power server through PowerVC***

Using the output that is shown in Example 5-16 on page 248, select which VMs that you want to display by filtering on items such as status, network, image, or hosted IBM Power server. You can select which values you display in your output.

In Example 5-17 on page 253, you use the `openstack.cloud.server_info` module to retrieve information about VMs on a certain IBM Power server, and display the name, status, and memory allocation of those VMs.

*Example 5-17 Displaying the name, status, and memory of all VMs on a specific IBM Power server*

---

```
- name: Collect information of all VMs on PowerServer1 through PowerVC
  openstack.cloud.server_info:
    cloud: powervc_cloud
    filters:
      compute_host: "{{ Server_serial_number }}"
    register: vm_on_host_results

- name: Show the name, status, and memory of VMs on PowerServer1
  debug:
    msg: "{{ vm_on_host_results | json_query('servers[*].
      {name: name, status: vm_state, memory: flavor.ram}') }}"
```

TASK [Show the name, status, and memory of VMs on PowerServer1]

\*\*\*\*\*

```
ok: [localhost] => {
  "msg": [
    {
      "memory": 4096,
      "name": "aix-vm-1",
      "status": "active"
    },
    {
      "memory": 4096,
      "name": "ibmi-vm-1",
      "status": "active"
    }
  ]
}
```

---

## Stopping or starting a PowerVC VM by using the openstack.cloud modules

Another useful module from the OpenStack Cloud collection is the `server_action` module. You can use this module to perform a stop or start action against an existing VM through PowerVC.

Example 5-18 shows an example of using `openstack.cloud.server_action`.

*Example 5-18 Playbook for stopping or starting a PowerVC VM*

---

```
- name:
  openstack.cloud.server_action:
    cloud: powervc_cloud
    name: "{{ VM_Name }}"
    action: <stop/start>
    register: result
```

---

## Creating and attaching a storage volume by using the openstack.cloud modules

Using the `openstack.cloud` modules, you can create and attach a new volume in IBM PowerVC. To do so, use two of the modules from the collection:

- ▶ `openstack.cloud.volume` to create the volume.
- ▶ `openstack.cloud.server_volume` to attach the volume to an existing VM.

The `openstack.cloud.volume` module documentation can be found at [Ansible Community Documentation](#).

Example 5-19 shows an example of using this module to create a 10 GB storage volume.

*Example 5-19 Creating a 10 GB storage volume by using the OpenStack Cloud collection*

---

```
- name: Create a {{ new_disk_size }}GB volume, called {{ new_disk_name }} using
storage template {{ storage_template }}
  openstack.cloud.volume:
    cloud: powervc_cloud
    state: present
    name: "{{ new_disk_name }}"
    size: "{{ new_disk_size }}"
    volume_type: "{{ storage_template }}"
    register: volume_create_information
```

---

**Note:** The size of the volume is in GB, and the `volume_type` refers to the PowerVC storage template to use.

After you create the new storage volume, you can attach it to an existing VM by using the OpenStack Cloud server volume module. The documentation for that module can be found at [Ansible Community Documentation](#).

Example 5-20 shows an example of using this module to attach the volume to an existing VM in PowerVC.

*Example 5-20 Attaching a storage volume to an existing PowerVC VM*

---

```
- name: "Attach storage volume {{ new_disk_name }} to VM {{ VM_Name }}"
  openstack.cloud.server_volume:
    cloud: powervc_cloud
    state: present
    server: "{{ VM_Name }}"
    volume: "{{ new_disk_name }}"
    register: volume_attach_information
```

---

You can combine both tasks in the same playbook to first create and then attach the new storage to an existing VM, as shown in Example 5-21 and Figure 5-3 on page 255.

*Example 5-21 Output showing creating and attaching a new storage volume through PowerVC*

---

```
PLAY [Connected to PowerVC/Openstack VM, create new disk and attach to VM]
*****

TASK [Create a 10 GB volume, called data_volume_1 by using storage template V7K1
Secondary Pool] *****
changed: [localhost]
```

```

TASK [Attach storage volume data_volume_1 to VM aix-vm-1]
*****
changed: [localhost]

PLAY RECAP
*****
localhost : ok=2    changed=2    unreachable=0    failed=0    skipped=0
rescued=0   ignored=0

```

The results are shown in the PowerVC UI, as shown in Figure 5-3.

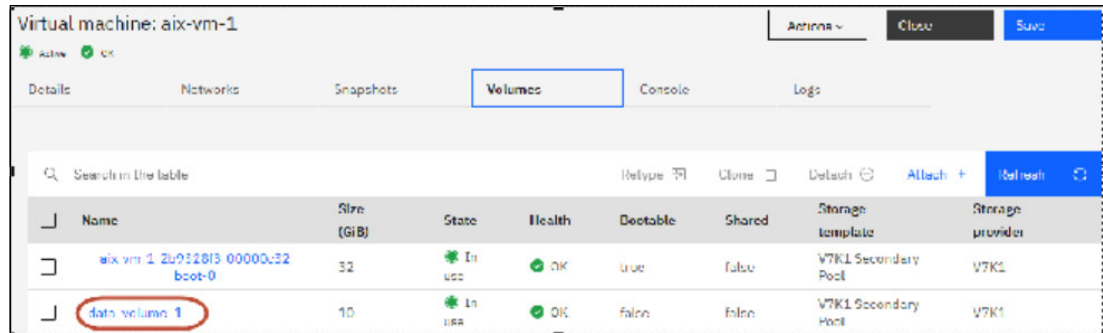


Figure 5-3 PowerVC UI showing a new volume that is attached to a VM

### 5.1.3 Using the URI modules to interact with PowerVC API services

Another method of automating IBM PowerVC by using Ansible is to use the REST APIs that IBM PowerVC provides. The OpenStack software has industry-standard interfaces that are released under the terms of the Apache License. IBM PowerVC interfaces are a subset of OpenStack northbound APIs.

This section describes the following IaC options by using the URI module that uses PowerVC API services:

- ▶ Authentication
- ▶ Creating a VM
- ▶ Destroying an existing VM
- ▶ Showing resource information
- ▶ Resizing an online VM

Several interfaces were added or extended to enhance the capabilities that are associated with the IBM Power platform REST APIs.

APIs use a common set of methods that you can use to perform operations on IBM PowerVC:

POST	Create operation
GET	Read operation
PUT	Update operation
DELETE	Delete operation

There are three types of APIs that you can use to integrate Ansible with PowerVC:

- Supported OpenStack APIs** These APIs are a subset of the APIs that provided by OpenStack and can be used with PowerVC without any modifications.
- Extended OpenStack APIs** These APIs are a subset of the APIs that provided by OpenStack, but their functions are extended by PowerVC.
- PowerVC APIs** These APIs do not exist in OpenStack and are exclusive to PowerVC.

PowerVC APIs are provided by several, specialized inter-operable services. Each service is accessible on a distinct port number and provides a set of APIs that run specialized functions that are related to that service. The services are shown in Table 5-2.

Table 5-2 PowerVC API services

Project or service name	Description
<b>OpenStack projects</b>	
Telemetry (Ceilometer)	Billing, benchmarking, scalability, and statistics. Used for auditing in PowerVC.
Storage (Cinder)	Storage and storage volume management.
Image (Glance)	Images and image management.
Identity (Keystone)	Security, identity, and authentication services.
Networking (Neutron)	Networking and network management.
Compute (Nova)	Host or compute. Manages the lifecycle and operations of compute resources.
<b>PowerVC services</b>	
Validator	Validates the PowerVC environment.

The OpenStack APIs that are shown in Table 5-2 can read, create, update, and delete IBM PowerVC resources, including VMs, networks, storage, key pairs, images, and projects. For more information, see the [OpenStack organization site](#).

The IBM PowerVC APIs (along with references to the OpenStack APIs) are documented at [PowerVC documentation](#).

Each OpenStack and IBM PowerVC API service uses a unique port. Some of the key API ports are shown in Table 5-3.

Table 5-3 PowerVC API service ports

Service	Function	Port
Keystone	Identify/authentication	5000
Nova	Compute	8774
Neutron	Network	9696
Glance	Images	9292
Cinder	Storage	9000



To access IBM PowerVC APIs through Ansible, use the `uri` module, which is part of `ansible-core` (`ansible.builtin.uri`).

## Authenticating with IBM PowerVC (API)

Before you perform any API actions on IBM PowerVC, obtain an authentication token. To do so, perform an API POST to the PowerVC server with the following information:

- ▶ API URL of the PowerVC server (IP or hostname)
- ▶ Keystone authentication port (default 5000) URI
- ▶ PowerVC username and password
- ▶ Tenant/Project name
- ▶ Domain name (Only default is supported.)

Example 5-22 shows Ansible URI module authenticating with IBM PowerVC, obtaining the information that is required, setting a fact to store the authorization token, and then displaying the token.

*Example 5-22 Obtaining the authorization token from PowerVC by using the uri module*

---

```
- name: Connect to PowerVC and collect an auth token
  uri:
    url: https://{{ powervc_host }}:{{ auth_port }}/v3/auth/tokens
    method: POST
    body: '{"auth":{"scope":{"project":{"domain":{"name":"Default"},
        "name":"ibm-default"}}},
        "identity":{"password":{"user":{"domain":{"name":"Default"},
            "password":"{{ PowerVC_password }}",
            "name":"{{ PowerVC_ID }}}"},
        "methods":["password"]}}}'
    body_format: json
    use_proxy: no
    validate_certs: no
    status_code: 201
  register: auth

- name: Set Auth Token
  set_fact:
    auth_token: "{{ auth.x_subject_token }}"

- name: Display Auth Token
  debug:
    var: auth_token
```

---

Although you would not normally display the token, we do it in this case to demonstrate that Ansible authenticated with the PowerVC server. The output is shown in Example 5-23.

*Example 5-23 Authorization token output*

---

```
- TASK [Connect to PowerVC and collect auth token]
*****
ok: [localhost]

TASK [Set Auth Token]
*****
ok: [localhost]

TASK [Display Auth Token]
*****
ok: [localhost] => {
  "auth_token":
  "gAAAAABk26Y92U1l1u0uFuXzmv7JdU1-st3SPkf_1wTTQRE2ssm8yATw6KRMU9vGHtIJHaT5ZHGk18cHLd
  zRwoLqQhLtByBhKEw96-pKBFmDOPfswTaJiTsRAmddRaqM18Y4b4ZbmFrESaTI4pzzZH2uHIEby0KhPSm7
  -Wn5A58gg2RAaOARY3MrgeIHVvpDrmKOD3G1qwHv1-GNPFwZaqkZzKWm9XXXXXXXXXXXXXXXXXXXXX" \

```

---

Now that you have the fact set (we called it 'auth\_token' in Example 5-23), you can perform API operations against your PowerVC environment by using Ansible.

### Creating VMs with IBM PowerVC by using the URI module

To create a VM by using the URI module (once you have your authentication key), provide some key values and unique IDs, which include the following ones:

- ▶ The ID of the PowerVC Project that you want to deploy the new VM in, for example, `ibm-default`.
- ▶ The ID of the PowerVC Image that you want to use for the new VM.
- ▶ The ID of the PowerVC compute flavor that you want to use for the new VM.
- ▶ The ID of the PowerVC network on which to place the new VM.

There are several other optional values that you can supply, including availability zone (the host group or the name of the server), `key_name` (the SSH key pair name) and `network` patched IP (the specific IP address). These values are detailed in the [OpenStack API compute \(nova\) documentation](#).

Example 5-24 shows how to build a new VM by using the URI module through the IBM PowerVC API nova service.

*Example 5-24 Creating a VM on PowerVC by using the URI module and API*

---

```
- name: Connect to PowerVC with token and create a new VM
  uri:
    url: https://{ powervc_host }:{ nova_port }/v2.1/{ project_id
  }}/servers
    method: POST
    use_proxy: no
    validate_certs: no
    return_content: no
    body: '{
      "server": {
        "name": "{ new_vm_name }",
        "imageRef": "{ image_UID }",

```

```

        "flavorRef": "{{ flavor_UID }}",
        "availability_zone": "{{ host_group_name }}",
        "networks": [{
            "uuid": "{{ network_UID }}"
        }]
    }
}'
body_format: json
headers:
    Accept: "application/json"
    Content-Type: "application/json"
    OpenStack-API-Version: "compute 2.46"
    User-Agent: "python-novaclient"
    X-Auth-Token: "{{ auth_token }}"
    X-OpenStack-Nova-API-Version: "2.46"
status_code: 202
register: vm_create

```

---

**Note:** In Example 5-24 on page 258, pass the project ID, image ID, flavor ID, and network ID.

The status code for a successful deployment is 202.

## Destroying VMs with IBM PowerVC by using the URI module

To destroy a VM by using API services from Power Systems Virtual Server, you must know two things:

- ▶ The ID of the PowerVC Project where the VM is hosted, for example, `ibm-default`.
- ▶ The ID of the PowerVC VM that you want to destroy.

**Note:** When you create a VM, you pass the VM name to the API; however, for an existing VM, use the VM's unique ID.

To learn how to retrieve a project ID, see “Collecting a project ID by using the project name from PowerVC by using the URI module” on page 260.

To learn how to retrieve a Virtual Server Instance (VSI) ID, see “Collecting a VM ID by using the VSI name from PowerVC by using the URI module” on page 261.

Example 5-25 shows how to destroy an existing VM by using the URI module through the IBM PowerVC API nova service.

### *Example 5-25 Destroying an existing VM on PowerVC by using the URI module and API*

```

- name: Connect to PowerVC with token and destroy a VM
  uri:
    url: https://{{ powervc_host }}:{{ nova_port }}/v2.1/{{ project_id }}/servers/{{
vm_id }}
    method: DELETE
    use_proxy: no
    validate_certs: no
    return_content: no
    headers:
      X-Auth-Token: "{{ auth_token }}"
    status_code: 204
  register: vm_destroy

```

---

**Note:** The status code for a successful VM destruction is 204.

## Retrieving resource information from IBM PowerVC by using the URI module

When using APIs, you cannot always refer to names such as the project name, VM name, network name, or others when referencing the endpoint. OpenStack and PowerVC APIs work with unique IDs. Although there are many of these IDs from the PowerVC UI or the OpenStack CLI, we do not expect people developing Ansible playbooks to know them or hardcode them. Therefore, you must convert resource names into their resource IDs before you can perform any meaningful operations on PowerVC by using APIs.

### ***Collecting a project ID by using the project name from PowerVC by using the URI module***

Example 5-26 shows the URI module connecting to the PowerVC projects API service to retrieve all project information by using the authorization token that was collected in Example 5-23 on page 258. Filter that information to select only the project that you are interested in (`ibm-default` in this case). Set a fact that is called `project_id` that contains only the ID of the selected project and display that ID.

*Example 5-26 Retrieving a PowerVC project ID by using the URI module*

---

```
- name: Connect to PowerVC with auth token to collect project information
  uri:
    url: https://{{ powervc_host }}:{{ auth_port }}/v3/projects
    method: GET
    use_proxy: no
    validate_certs: no
    return_content: no
    headers:
      X-Auth-Token: "{{ auth_token }}"
  register: project_information

- name: Collect ID of chosen project in array format
  set_fact:
    project_id_array: "{{ project_information.json | json_query(query) }}"
  vars:
    query: "projects[?name=='ibm-default'}.{id: id}"

- name: Collect project ID for selected project
  set_fact:
    project_id: "{{ project_id_array.0['id'] }}"

- name: Show Project ID
  debug:
    var: project_id
```

---

The output from Example 5-26 is shown in Example 5-27 on page 261.

*Example 5-27 Output from using the URI module to retrieve a project ID from a project name*

---

```
TASK [Collect ID of chosen project in array format]
*****
ok: [localhost]

TASK [Collect project ID for selected project]
*****
ok: [localhost]

TASK [Show Project ID]
*****
ok: [localhost] => {
  "project_id": "6a01a6c6f13c40f79b7ff5552170a371"
}
```

---

Now, you can use that project ID variable in future PowerVC API Ansible playbooks, such as creating a VM.

***Collecting a VM ID by using the VSI name from PowerVC by using the URI module***

Example 5-28 shows the URI module connecting to the PowerVC nova API service to retrieve information about all the VMs (servers) by using the authorization token that was collected in Example 5-26 on page 260 and the project ID that was collected in Example 5-27. Filter that information to select only the VM that you are interested in. Set a fact called `vm_id` that contains only the ID of the selected VM and display that ID.

*Example 5-28 Retrieving a PowerVC VM ID by using the URI module*

---

```
- name: Connect to PowerVC with auth token and project ID to collect VM
information
  uri:
    url: https://{{ powervc_host }}:{{ nova_port }}/v2.1/{{ project_id
}}/servers
    method: GET
    use_proxy: no
    validate_certs: no
    return_content: no
    headers:
      X-Auth-Token: "{{ auth_token }}"
    register: vm_information

- name: Collect ID of chosen VM in array format
  set_fact:
    vm_id_array: "{{ vm_information.json | json_query(query) }}"
  vars:
    query: "servers[?name=='{{ vm_name }}'].{id: id}"

- name: Collect VM ID for selected VM
  set_fact:
    vm_id: "{{ vm_id_array.0['id'] }}"

- name: Show VM ID
  debug:
    var: vm_id
```

---

**Note:** In Example 5-28, we had to use the `project_id` in the API URL, and we passed the VM name as variable `<{{ vm_name }}>`.

The output from Example 5-28 on page 261 is shown in Example 5-29.

*Example 5-29 Output from using the URI module to retrieve a VM ID from a VM name*

---

```
TASK [Collect ID of chosen VM in array format]
*****
ok: [localhost]

TASK [Collect VM ID for selected VM]
*****
ok: [localhost]

TASK [Show VM ID]
*****
ok: [localhost] => {
    "vm_id": "1b9efb52-3b8a-4927-af84-c0feef495c1f"
}
```

---

Now, you can use that VM ID variable in future PowerVC API Ansible playbooks, such as destroying an existing VM or performing PowerVC operations against that VM.

## Resizing a VM by using URI modules and PowerVC API services

A key advantage of using the PowerVC API services is that you can perform more detailed tasks such as resizing a VM. This task can be useful if a VM is low on CPU or memory resources and you want to increase them, or when a VM must reduce its resources, for example, after a development test phase.

This section introduces the VM `action` API service that you can use to perform several different actions against an existing VM, including online resizing. The options are documented in the [IBM PowerVC documentation](#).

In Example 5-30, you pass the VM `action` API service the new values for the required CPU and memory.

*Example 5-30 Resizing an active VM by using the URI module and PowerVC API services*

---

```
- name: "Connect to PowerVC with token and resize VM {{ vm_name }} to {{
new_total_proc_units }} processors, and {{ new_total_memory_mb }}MB"
  uri:
    url: https://{{ powervc_host }}:{{ nova_port }}/v2.1/{{ project_id
}}/servers/{{ vm_id }}/action
    method: POST
    use_proxy: no
    validate_certs: no
    return_content: no
    body: {
      "resize": {
        "flavor": {
          "vcpus": "{{ new_vcpus }}",
          "disk": "0",
          "extra_specs": {
            "powervm:proc_units": "{{ new_total_proc_units }}",
```

```

    },
    "ram": "{{ new_total_memory_mb }}"
  }
}
}
body_format: json
headers:
  Accept: "application/json"
  Content-Type: "application/json"
  OpenStack-API-Version: "compute 2.46"
  User-Agent: "python-novaclient"
  X-Auth-Token: "{{ auth_token }}"
  X-OpenStack-Nova-API-Version: "2.46"
status_code: 202
register: vm_resize_details

```

The VM that you created in “Creating VMs with IBM PowerVC by using the URI module” on page 258 was assigned one vCPU, 0.5 entitled cores, and 4 GB of memory, as shown in Figure 5-4.



Figure 5-4 PowerVC UI showing a VM resource before resizing by using API services

Example 5-31 shows the output from the resized playbook.

*Example 5-31 Output of resizing of an online VM by using the URI module and PowerVC API services*

```

TASK [Show current CPU and memory allocation for VM aix-vm-1]
*****
ok: [localhost] => {
  "current_vm_spec_details": {
    "CPUs": "0.50",
    "Memory": 4096,
    "name": "aix-vm-1",
    "vCPUs": 1
  }
}

TASK [Connect to PowerVC and resize VM aix-vm-1 to 0.75 procesors, and 6144 MB]
*****
ok: [localhost]

TASK [Connect to PowerVC with token and wait for VM aix-vm-1 to be in state
'VERIFY_RESIZE'] *****
FAILED - RETRYING: [localhost]: Connect to PowerVC with token and wait for VM
aix-vm-1 to be in state 'VERIFY_RESIZE' (6 retries left).
FAILED - RETRYING: [localhost]: Connect to PowerVC with token and wait for VM
aix-vm-1 to be in state 'VERIFY_RESIZE' (5 retries left).

```

```

ok: [localhost]

TASK [Connect to PowerVC with token and confirm resize of VM aix-vm-1]
*****
ok: [localhost]

TASK [Pause for 30 seconds to allow resizing to complete]
*****
Pausing for 30 seconds
(Ctrl+C then 'C' = continue early, Ctrl+C then 'A' = abort)
ok: [localhost]

TASK [Connect to PowerVC and collect new CPU and memory information for VM
aix-vm-1 after the resize]
*****
ok: [localhost]

TASK [Show new CPU and memory allocation for VM aix-vm-1]
*****
ok: [localhost] => {
  "new_vm_spec_details": {
    "CPUs": "0.75",
    "Memory": 6144,
    "name": "aix-vm-1",
    "vCPUs": 1
  }
}

PLAY RECAP
*****
localhost : ok=19   changed=0    unreachable=0    failed=0    skipped=6
rescued=0   ignored=0

```

The output that is shown in Example 5-31 on page 263 shows that the VM reported 0.5 CPU entitlement and 4 GB of memory before the resize and 0.75 CPU entitlement and 6 GB of memory after the resize.

Figure 5-5 shows the resize being performed.

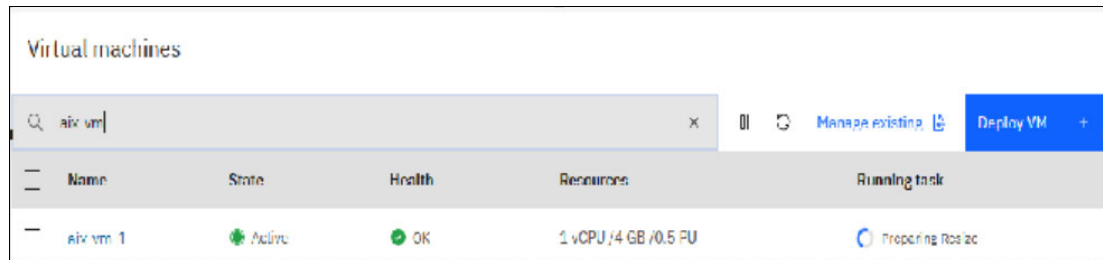


Figure 5-5 PowerVC UI showing a VM resizing by using API services



After the resize completes, you can verify the results in the PowerVC UI, as shown in Figure 5-6.



Figure 5-6 PowerVC UI showing VM after resizing by using API services

## 5.2 IBM Power Systems Virtual Server

IBM Power Systems Virtual Server is an Infrastructure as a Service (IaaS) offering that customers can use to deploy AIX, IBM i, and Linux workloads in a public cloud environment. The Power Systems Virtual Server data centers are spread across the globe, including America, Canada, Brazil, UK, Germany, Japan, and Australia.

**Note:** Within Power Systems Virtual Server, VMs are referred to as VSIs.

### 5.2.1 Using the IBM Cloud collection for Power Systems Virtual Server

IBM created a collection (`ibm.cloudcollection`) so that Ansible can interact with IBM Cloud. This collection includes several Power infrastructure modules for use within IBM Power Systems Virtual Server. The collection is available on Ansible Galaxy [here](#).

This collection can be installed as shown in Example 5-32.

*Example 5-32 Installing the `ibm.cloudcollection` from Ansible Galaxy*

```
# ansible-galaxy collection install ibm.cloudcollection
Process install dependency map
Starting collection install process
Installing 'ibm.cloudcollection:1.49.0' to
'/root/.ansible/collections/ansible_collections/ibm/cloudcollection'
```

Several key PI modules are shown in Table 5-4

Table 5-4 IBM Cloud Collection PI modules

Module name	Function
<code>ibm_pi_catalog_images_info</code>	Collects information about Power Systems Virtual Server catalog images.
<code>ibm_pi_cloud_connection</code>	Creates, updates, or destroys an IBM Cloud connection.
<code>ibm_pi_cloud_instance_info</code>	Collects information about a Power Systems Virtual Server service instance.
<code>ibm_pi_instance</code>	Creates, updates, or destroys a VSI.
<code>ibm_pi_instance_action</code>	Performs an action against a VSI.

Module name	Function
ibm_pi_instance_info	Collects information about a VSI.
ibm_pi_instances_info	Collects information about all VSIs.
ibm_pi_network	Creates, updates, or destroys a Power Systems Virtual Server network.
ibm_pi_volume	Creates, updates, or destroys a Power Systems Virtual Server storage volume.
ibm_pi_volume_attach	Attaches a Power Systems Virtual Server storage volume to a VSI.

In total, there are over 70 Power Systems Virtual Server specific modules in the collection.

**Note:** The IBM Cloud Power Systems Virtual Server modules generate Terraform code to perform actions against the Power Systems Virtual Server API services. At the time of writing, you must have Terraform 0.10.20 installed. The Terraform resources and data sources that they call can be found at the [Terraform Registry](#).

## Creating a VSI in Power Systems Virtual Server by using the IBM Cloud Collection

This section shows how to create a VSI by using the `ibm.cloudcollection.ibm.pi_instance` module. When you use the IBM Cloud collection, you pass the API key, the cloud instance and resource IDs, and the region in each task to the modules.

Example 5-33 shows the creation of a VSI.

*Example 5-33 Creating a VSI by using the IBM Cloud Collection module `ibm_pi_instance`*

---

```
- name: Create a POWER Virtual Server Instance
  ibm.cloudcollection.ibm_pi_instance:
    state: available
    pi_cloud_instance_id: "{{ pi_cloud_instance_id }}"
    ibmcloud_api_key: "{{ ibmcloud_api_key }}"
    id: "{{ pi_instance.resource.id | default(omit) }}"
    region: "{{ region }}"
    pi_memory: "{{ memory }}"
    pi_processors: "{{ processors }}"
    pi_instance_name: "{{ vsi_name }}"
    pi_proc_type: "{{ proc_type }}"
    pi_image_id: "{{ image_dict[image_name_to_be_created] }}"
    pi_volume_ids: []
    pi_network_ids:
      - "{{ pi_network.id }}"
    pi_key_pair_name: "{{ pi_ssh_key.pi_key_name }}"
    pi_sys_type: "{{ sys_type }}"
    pi_replication_policy: none
    pi_replication_scheme: sufpatch
    pi_replicants: "1"
    pi_storage_type: "{{ disk_type }}"
    register: pi_instance_create_output
```

---

**Note:** The state option for the `ibm_pi_instance` module, which helps ensure that a VSI exists, is available.

## Destroying a VSI in Power Systems Virtual Server by using the IBM Cloud Collection

To destroy a VSI by using the `ibm.cloudcollection.ibm_pi_instance` module, define the state of that VSI as absent, as shown in Example 5-34.

*Example 5-34 Destroying a VSI by using the IBM cloud collection `ibm_pi_instance` module*

---

```
- name: Destory a POWER Virtual Server Instance
  ibm.cloudcollection.ibm_pi_instance:
    state: absent
    pi_cloud_instance_id: "{{ pi_cloud_instance_id }}"
    ibmcloud_api_key: "{{ ibmcloud_api_key }}"
    id: "{{ pi_instance.resource.id | default(omit) }}"
    region: "{{ region }}"
  register: pi_instance_destroy_output
```

---

## 5.2.2 Using the URI module for Power Systems Virtual Server

You can use the Ansible URI module to make calls to API services in IBM Cloud Power Systems Virtual Server, similar to OpenStack and PowerVC, as described in 5.1.3, “Using the URI modules to interact with PowerVC API services” on page 255.

Like OpenStack, IBM Power Systems Virtual Server has many API services that you can use to manage resources, such as VSIs, images, storage volumes, key pairs, networks, snapshots, VPNs, and others. These APIs are documented in the [IBM Cloud documentation](#).

Power Systems Virtual Server services use regional endpoints over both public and private networks. To target the public service, replace `{region}` with the prepatch that represents the geographic area where the public facing service is located in the URL that is shown in Example 5-35. At the time of writing, these locations are `us-east` (Washington DC), `us-south` (Dallas, Texas), `eu-de` (Frankfurt, Germany), `lon` (London, UK), `tor` (Toronto, Canada), `syd` (Sydney, Australia), and `tok` (Tokyo, Japan).

*Example 5-35 Public regional endpoint for IBM Power Systems Virtual Server*

---

```
https://{region}.power-iaas.cloud.ibm.com
```

---

To target the private service, you need to replace `{region}` with the prepatch that represents the geographic area where the private facing service is located in the URL shown in Example 5-36. At the time of writing, these data centers are `us-east` (Washington DC), `us-south` (Dallas, Texas), `eu-de` (Frankfurt, Germany), `eu-gb` (London, UK), `ca-tor` (Toronto, Canada), `au-syd` (Sydney, Australia), `jp-tok` (Tokyo, Japan), `jp-osa` (Osaka, Japan), `br-sao` (Sao Paulo, Brazil), and `ca-mon` (Montreal, Canada).

*Example 5-36 Private regional endpoint for IBM Power Systems Virtual Server*

---

```
https://private.{region}.power-iaas.cloud.ibm.com
```

---

All the IBM Cloud Power Systems Virtual Server API methods are also documented, along with the API service URL, the required parameters, and the response body. For example, to obtain information about all the VSIs, see [Get all the pvm instances for this cloud instance](#).

Example 5-37 shows an example request to retrieve all VSIs within IBM Power Systems Virtual Server.

*Example 5-37 Example request to get all Power Systems Virtual Server VSI information*

---

```
curl -X GET
https://{region}.power-iaas.cloud.ibm.com/pcloud/v1/cloud-instances/${CLOUD_INSTANCE_ID}/pvm-instances
-H 'Authorization: Bearer <>'
-H 'CRN: crn:v1...'
-H 'Content-Type: application/json'
```

---

## Authenticating with Power Systems Virtual Server by using APIs

Before you can perform any action against the API services that are presented by IBM Power Systems Virtual Server, you must first authenticate. To do this task, you need an IBM Cloud API key, which you use to obtain a Cloud IAM access token. This IAM access token (often referred to as an auth token) can be used to directly access the API services.

IBM Cloud API keys are associated with a user's identity and can be used to access cloud platform and APIs, depending on the access that is assigned to the user. The API access key is created by using the process that is described at [IBM Cloud](#).

**Note:** When creating an IBM Cloud API key, record the key in a safe location because you cannot retrieve the contents after the key is created.

Example 5-38 shows how to obtain the auth token by using the URI module along with the IBM Cloud API key.

*Example 5-38 Obtaining the IAM access token (auth token) by using the URI module and IBM Cloud API key*

---

```
- name: Obtain IBM Cloud Power Systems Virtual Server authorization token by using
an IBM Cloud API key
  hosts: localhost
  gather_facts: no

  vars:
    - auth_data: "grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey="
      api_key: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"

  tasks:
    - name: Get an IAM access token
      uri:
        url: "https://iam.cloud.ibm.com/identity/token"
        method: POST
        force_basic_auth: true
        validate_certs: yes
        headers:
          content-type: "application/x-www-form-urlencoded"
          accept: "application/json"
        body: "{{ auth_data }}{{ api_key|trim }}"
```

```

    body_format: json
    register: iam_token_request

- name: Set auth token fact
  set_fact:
    auth_token: "{{ iam_token_request.json.access_token }}"

- name: Show token
  debug:
    var: auth_token

```

---

Example 5-39 shows the IBM Cloud ID, and in the body of the API POST, you pass two values:

- ▶ Authorization Data (`grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey=`)
- ▶ IBM API key (trimmed)

The last task in the playbook outputs the `auth_token` fact, which you populated. Normally, this fact is hidden, but we included it so that we can confirm it was created correctly.

*Example 5-39 Output from an IAM access token retrieval*

---

```

PLAY [Obtain IBM Cloud PowerVC auth token by using API]
*****

TASK [Get IAM access token]
*****
ok: [localhost]

TASK [Show token]
*****
ok: [localhost] => {
  "auth_token": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}

```

---

This `auth_token` can now be used within the Ansible playbook to perform actions against the IBM Power Systems Virtual Server API services.

Before you perform an action against the IBM Power Systems Virtual Server API services that is associated with your IBM Power resources, you must know the Power Systems VSI ID (also known as the Cloud Resource Name (CRN)). You can obtain the CRN by using the `ibmcloud` command (as shown in Example 5-40) or by using the IBM Cloud UI.

*Example 5-40 Obtaining the Cloud Resource Name by using the `ibmcloud` command*

---

```

% ibmcloud resource service-instance "Power Virtual Server-London 06" --id
Retrieving service instance Power Virtual Server-London 06 in all resource groups
under account XXX YYYYY's Account as x_yyyyy@uk.ibm.com...
crn:v1:bluemix:public:power-iaas:lon06:a/abcdefghijklmnopqrstuvwxyabcdef:121d5ee5
-b87d-4a0e-86b8-aaff422135478: :

```

---

In Example 5-40, the CRN includes the following items:

- ▶ Tenant ID, which in the example is `abcdefghijklmnopqrstuvwxyabcdef`.
- ▶ Cloud Instance ID, which in the example is `121d5ee5-b87d-4a0e-86b8-aaff422135478`.

In addition to defining the CRN tenant ID and cloud instance ID, you must define the CRN values that are shown in Example 5-41.

*Example 5-41 CRN values that are required to connect to Power Systems Virtual Server London 04 API services*

---

```
crn:
  version: "v1"
  cname: "bluemix"
  ctype: "public"
  service_name: "power-iaas"
  location: "lon04"
  tenant_id: "abcdefghijklnopqrstuvwxyzaef"
  cloud_instance_id: "121d5ee5-b87d-4a0e-86b8-aaff422135478"
```

---

## Retrieving resource information from IBM Power Systems Virtual Server by using the URI module

This section shows examples about how to retrieve information about resources such as VSIs, images, and networks from IBM Power Systems Virtual Server by using URI and API servers.

### *Retrieving information about all VSIs*

In Example 5-42, you retrieve the names of all existing VSIs in your Power Systems Virtual Server workspace by using the CRN values that are defined, along with the GET method and the URL that is documented in Example 5-38 on page 268.

*Example 5-42 Retrieving all VSI names in Power Systems Virtual Server by using the URI module and APIs*

---

```
- name: Collect information about all the VSIs in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
    }}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{
      crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
      crn.cloud_instance_id }}:"
      Content-Type: application/json
    register: pvs_existing_vsi_results

- name: Set VSI list of names
  set_fact:
    vsi_names: "{{ vsi_names | default([]) + [item] }}"
  with_items: "{{ pvs_existing_vsi_results | json_query(query_to_run) }}"
  vars:
    query_to_run: 'json.pvmInstances[*].serverName'
```

---

**Note:** In the CRN values that shown in Example 5-42, the tenant ID and cloud instance ID are the ones that were collected in Example 5-41 on page 270. Location must be one of the locations that are listed in the 'ibmcloud catalog locations' CLI output, that is, fra01, fra02, lon04, lon06, dal10, dal12, wdc06, wdc07, mon01, tor01, osa21, sao01, sao04, syd04, syd05, or tok04.

You can see the output of the VSI retrieval request in Example 5-43.

*Example 5-43 Displaying the names of all Power Systems Virtual Server VSIs by using the URI module and APIs*

---

```
TASK [Collect information about all the VSIs in this cloud instance]
*****
ok: [localhost]

TASK [Set VSI list of names]
*****
*****
ok: [localhost] => (item=aix-vsi-1)
ok: [localhost] => (item=ibmi-vsi-1)
```

---

### **Retrieving information about all images**

In Example 5-44, you retrieve the names of all existing images in your Power Systems Virtual Server catalog. The API syntax can be found at [Power Cloud API](#).

*Example 5-44 Retrieving all VSI images within the Power Systems Virtual Server environment by using the URI module and APIs*

---

```
- name: Collect information about all the images in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/images"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{
crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
crn.cloud_instance_id }}:"
      Content-Type: application/json
    register: pvs_images_results

- name: Set a list of image names
  set_fact:
    image_names: "{{ image_names | default([]) + [item] }}"
  with_items: "{{ pvs_images_results | json_query(query_to_run) }}"
  vars:
    query_to_run: 'json.images[*].name'

- name: Show all image names
  debug:
    var: image_names
```

---

Example 5-45 shows the output of the image retrieval.

*Example 5-45 Showing all images within the Power Systems Virtual Server environment*

---

```
TASK [Set VSI list of names]
*****
ok: [localhost] => (item=7300-01-01)
ok: [localhost] => (item=IBMi-75-01-2984-2)

TASK [Show all image names]
*****
```

```
ok: [localhost] => {
  "image_names": [
    "7300-01-01",
    "IBMi-75-01-2984-2"
  ]
}
```

---

### **Retrieving information about all networks**

In Example 5-46, you retrieve the names of all existing images in your Power Systems Virtual Server environment. The API syntax is documented at [Power Cloud API](#).

*Example 5-46 Retrieving all networks within the Power Systems Virtual Server environment by using the URI module and APIs*

---

```
- name: Collect information about all the networks within the Power Systems
Virtual Server environment
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/networks"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{
crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_network_results

- name: Set network of names
  set_fact:
    network_names: "{{ network_names | default([]) + [item] }}"
  with_items: "{{ pvs_network_results | json_query(query_to_run) }}"
  vars:
    query_to_run: 'json.networks[*].name'

- name: Show all network names
  debug:
    var: network_names
```

---

Example 5-47 shows the output of the network retrieval.

*Example 5-47 Showing all networks within the Power Systems Virtual Server environment*

---

```
TASK [Set network of names]
*****
ok: [localhost] => (item=public-192_168_151_128-29-VLAN_2044)
ok: [localhost] => (item=private-subnet2)
ok: [localhost] => (item=private-subnet1)

TASK [Show all network names]
*****
ok: [localhost] => {
  "network_names": [
    "public-192_168_151_128-29-VLAN_2044",
    "private-subnet2",
    "private-subnet1"
```



```
]
}
```

---

### **Retrieving all information about a specific resource in Power Systems Virtual Server**

As with OpenStack, when using IBM Power Systems Virtual Server APIs, you cannot always refer to names such as VSI name, image name, network name, or others when referencing the endpoint. When you want to perform an action against a specific resource, you must refer to the resources unique ID. Therefore, you must first retrieve that ID before you can reference it. For example, if you want to show all the details about a specific VSI, you must provide the ID of that VSI. To do this task, you convert the VSI name (which you know) into its ID.

In Example 5-48, you use the URI module and the Power Systems Virtual Server `pvm-instances` API service to retrieve information about all the VSIs. Then, you filter that information and collect only the ID of the VM that you are interested in `{{ vsi_name }}`.

---

#### *Example 5-48 Retrieving the ID of a Power Systems Virtual Server VSI by using its name*

---

```
- name: Collect information about all the VSIs in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
  }}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{
crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_existing_vsi_results

- name: Collect ID of chosen VSI in array format
  set_fact:
    vsi_id_array: "{{ pvs_existing_vsi_results.json | json_query(query) }}"
  vars:
    query: "pvmInstances[?serverName=='{{ vsi_name }}'].{id: pvmInstanceID}"

- name: Collect VSI ID for selected VM
  set_fact:
    vsi_id: "{{ vsi_id_array.0['id'] }}"

- name: Show VSI ID
  debug:
    msg: "ID for {{ vsi_name }} is: {{ vsi_id }}"
```

---

**Note:** The VSI ID is called `pvmInstanceID` in Power Systems Virtual Server.

Example 5-49 shows the output from the VSI ID retrieval.

*Example 5-49 Showing the VSI ID*

---

```
TASK [Show VSI ID]
*****
ok: [localhost] => {
  "msg": "ID for ibmi-vsi-1 is: 2d8bf009-5922-40f4-9eef-d06fec7xxxxx"
}
```

---

## Creating a VSI in IBM Power Systems Virtual Server by using the URI module and API services

In this section, you create a Power Systems Virtual Server VSI by using the authorization token, along with the image name and network name that were collected in previous sections.

The syntax to create a VSI by using a POST HTTP method is documented at [Create a new PowerVM Instance](#).

Example 5-24 on page 258 showed an example where the content of the URI POST was contained within the body. In Example 5-50, you define a variable that is called `{{ vsi_info }}`, which contains all the required information, such as VSI name, image, network, and others. Then, you use that variable in the body section of the URI module.

*Example 5-50 Creating a VSI within Power Systems Virtual Server by using the RI module and API services*

---

Variable definition

```
vsi_info:
  serverName: "aix-vsi-1"
  imageID: "7300-01-01"
  processors: 1
  procType: "shared"
  memory: 4
  sysType: "s922"
  storageType: "tier3"
  networkIDs:
    - "public-192_168_xxx_xxx-VLAN_2044"
```

Create VSI task

```
- name: Create a VSI
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version }}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: POST
    status_code: 201
    body_format: json
    body: "{{vsi_info|to_json}}"
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name }}:{{ crn.location }}:a:{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}:"
      Content-Type: application/json
  register: pvs_create_vsi_result
```

---

**Note:** The status code for a successful VSI creation is 201.

Figure 5-7 shows the VSI being built in the Power Systems Virtual Server UI.

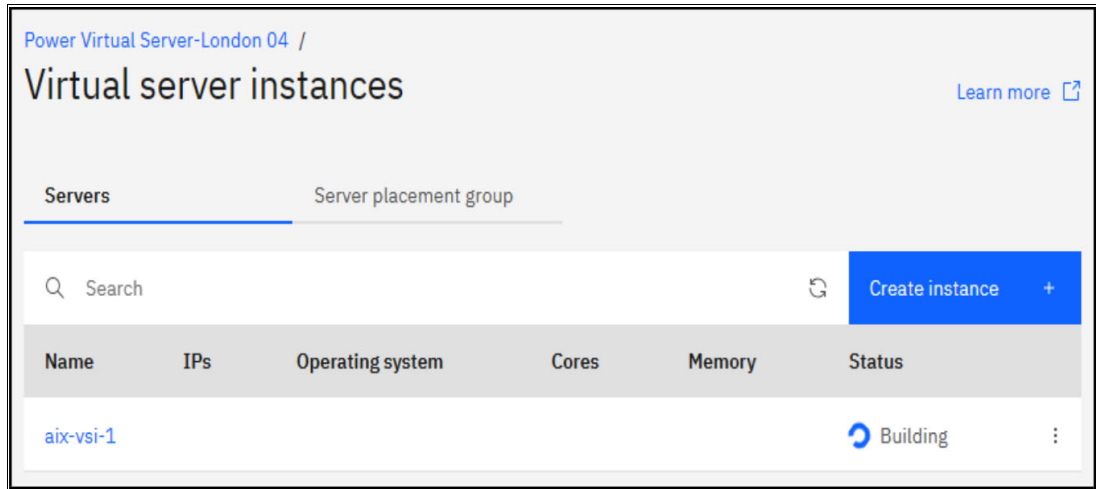


Figure 5-7 Power Systems Virtual Server UI: VSI building

## Destroying a VSI in IBM Power Systems Virtual Server by using the URI module and API services

In this section, you destroy an existing Power Systems Virtual Server VSI by using the authorization token and the VSI name. To destroy an existing VSI, pass its ID along with the DELETE HTTP method. Because you know only the VSI name, you must obtain its ID first.

The syntax to destroy a VSI by using a DELETE HTTP method is documented at [Delete a PCloud PVM Instance](#).

Example 5-51 shows how to destroy a VSI in Power Systems Virtual Server by passing the VSI name `{{ vsi_name }}`, which is then converted into the VSI ID that is used for the deletion command.

*Example 5-51 Destroying a Power Systems Virtual Server VSI by using the URI module and API*

```
- name: Collect information about all the VSIs in this cloud instance
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances"
    method: GET
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{
crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
crn.cloud_instance_id }}::"
      Content-Type: application/json
    register: pvs_existing_vsi_results

- name: Show all
  debug:
    var: pvs_existing_vsi_results

- name: Collect ID of chosen VSI in array format
```

```

set_fact:
  vsi_id_array: "{{ pvs_existing_vsi_results.json | json_query(query) }}"
vars:
  query: "pvmInstances[?serverName=='{{ vsi_name }}'].{id: pvmInstanceID}"

- name: Collect VSI ID for selected VM
  set_fact:
    vsi_id: "{{ vsi_id_array.0['id'] }}"

- name: Show details of VSI to destroy
  debug:
    msg: "Destroying VSI name: {{ vsi_name }} ID: {{ vsi_id }}"

- name: Destroying VSI
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances/{{ vsi_id }}"
    method: DELETE
    status_code: 200
    body_format: json
    #body: "{{ vsi_name }}"
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{
crn.service_name }}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{
crn.cloud_instance_id }}::"Content-Type: application/json
      register: pvs_destroy_vsi_result

```

The output from running the playbook that is shown in Example 5-51 on page 275 is shown in Example 5-52, where the name and ID of the VSI are displayed before they are destroyed.

*Example 5-52 Output of destroying a VSI in Power Systems Virtual Server by using the URI module and API*

```

TASK [Collect ID of chosen VSI in array format]
*****
ok: [localhost]

TASK [Collect VSI ID for selected VM]
*****
ok: [localhost]

TASK [Show details of VSI to destroy]
*****
ok: [localhost] => {
  "msg": "Destroying VSI name: aix-vsi-1 ID: d53fbe08-a613-41de-9016-047xxxxx"
}

TASK [Destroying VSI]
*****
ok: [localhost]

```

Figure 5-8 on page 277 shows the VSI deletion tasks that are recorded in the event logs on the Power Systems Virtual Server UI.

Resource type	Action	Date	Severity
Virtual Server Instance	Delete	8/22/2023, 6:45:47 AM	Info
Message: Power virtual server instance 'aix-vsi-1' has been deleted. User ID: IBMid-1100008F33 User email:			

Figure 5-8 Power Systems Virtual Server UI Event log showing VSI deletion

## Resizing a VSI on Power Systems Virtual Server by using APIs

By using the API services in Power Systems Virtual Server, you can resize an active VM by using the URI module within Ansible, which can be useful if resources become constrained on a VSI or if you want to reduce resources to save operational costs. In “Creating a VSI in IBM Power Systems Virtual Server by using the URI module and API services” on page 274, you built a VSI with 0.5 cores and 4 GB of memory, as shown in Figure 5-9.

Name	IPs	Operating system	Cores	Memory	Status
aix-vsi-1		AIX	0.5 cores	4 GiB	Active

Figure 5-9 IBM Power Systems Virtual Server before API resizing

Example 5-53 shows an example of using the URI module and API services to resize that VSI to 0.75 cores and 6 GB of memory.

Example 5-53 Resizing a Power Systems Virtual Server VSI by using a URI module and API services

```
- name: "Resize VSI {{ vsi_name }} to 0.75 cores and 6 GB of memory"
  uri:
    url: "https://{{ region }}.power-iaas.cloud.ibm.com/pcloud/{{ api_version
}}/cloud-instances/{{ crn.cloud_instance_id }}/pvm-instances/{{ vsi_id }}"
    method: PUT
    status_code: 202
    body_format: json
    body: '{
      "processors": 0.75,
      "memory": 6
    }'
    headers:
      Authorization: "Bearer {{ auth_token }}"
      CRN: "crn:{{ crn.version }}:{{ crn.cname }}:{{ crn.ctype }}:{{ crn.service_name
}}:{{ crn.location }}:a/{{ crn.tenant_id }}:{{ crn.cloud_instance_id }}::"
      Content-Type: application/json
  register: vsi_resize_details
```

During the resizing, you can see the status change on the Power Systems Virtual Server UI, as shown in Figure 5-10.

Name	IPs	Operating system	Cores	Memory	Status
aix-vsi-1		AIX	0.5 cores	4 GiB	Resize

Figure 5-10 Power Systems Virtual Server VSI resize

The output from the playbook also shows the resize taking place, as shown in Example 5-54.

*Example 5-54 Output of Power Systems Virtual Server VSI resizing by using API services*

---

```
TASK [Show existing CPU and memory allocation for VSI aix-vsi-1]
*****
ok: [localhost] => {
  "current_vsi_spec_details": {
    "CPUs": 0.5,
    "Memory": 4,
    "name": "aix-vsi-1",
    "vCPUs": 1
  }
}

TASK [Resize VSI aix-vsi-1 to 0.75 cores and 6 GB of memory]
*****
ok: [localhost]

TASK [Sleep to allow resize to complete]
*****
Pausing for 180 seconds
(Ctrl+C then 'C' = continue early, Ctrl+C then 'A' = abort)
ok: [localhost]

TASK [Show new CPU and memory allocation for VSI aix-vsi-1]
*****
ok: [localhost] => {
  "new_vsi_spec_details": {
    "CPUs": 0.75,
    "Memory": 6,
    "name": "aix-vsi-1",
    "vCPUs": 1
  }
}
PLAY RECAP
*****
localhost : ok=16  changed=0    unreachable=0    failed=0    skipped=2
rescued=0   ignored=0
```

---

The output from the Power Systems Virtual Server UI confirms that the resizing was successful, as shown in Figure 5-11.

Name	IPs	Operating system	Cores	Memory	Status
<a href="#">aix-vsi-1</a>	192.168.1.100	AIX	0.75 cores	6 GiB	Active

*Figure 5-11 Power Systems Virtual Server UI showing VSI post resizing by using API services*



## Day 2 management operations

What are “Day 2 operations”? This chapter defines what Day 2 operations are, and describes how you can use Ansible as a tool to help you automate those operations in your IBM Power environment, whether you are running Linux, AIX, or IBM i environments, and show you how you can optimize managing your hardware and software after the initial installation is done.

The following topics are described in this chapter:

- ▶ Introducing Day 2 operations
- ▶ Day 2 operations in Linux servers
- ▶ Day 2 operations in AIX environments
- ▶ Day 2 operations in IBM i environments

## 6.1 Introducing Day 2 operations

Day 2 operations are a group of tasks that monitor and maintain your environment after the initial installation. To use a technology, enterprises must understand how it fits into their broader architecture. Leaders must consider both their technical and operational architecture. After all, all systems are made up of software, people, and processes.

There are three stages of operations: Day 0, Day 1, and Day 2.

- ▶ Day 0 is the “design” stage, where you figure out what resources are required to provide the necessary functions for your information technology project.
- ▶ Day 1 operations describe the “deployment” stage, where you install, set up, and configure your environment.
- ▶ Day 2 is the “maintenance” stage. Typical Day 2 operations are focused on maintaining, monitoring, and optimizing the system. Day 2 operations continue throughout the product lifecycle because the system behavior must be continuously analyzed and patched.

In addition to monitoring how your environment is running, Day 2 operations also include routine tasks such as installing upgrades and updating systems.

Day 2 operations involve maintaining the products and platforms. They continuously monitor the health of the system, validate that it is meeting business requirements, track and patch issues that arise, and validate that all required patches and updates are applied to keep the environment secure. Because most of the Day 2 operations are continuous and repetitive activities, often they are tasks that should be considered for automation by using Ansible. For more information about how Red Hat supports Day 2 operations, see this [Red Hat Document](#).

The following sections describe using Ansible automation in your IBM Power environment (whether you use IBM AIX, IBM i, or Linux on Power) to help you manage the following functions within your environment:

- ▶ Storage
- ▶ Security and compliance
- ▶ Patches or upgrades
- ▶ Configuration and tuning

### 6.1.1 Storage

The storage tasks describe how to manage and maintain how your data is stored in your servers. These tasks involve things like monitoring file systems to help ensure that they do not run out of space, monitoring the performance of the storage to help ensure it is meeting business requirements, and managing a logical volume manager (LVM) and local file systems at the operating systems (OS) level. Using Ansible, you can automate the storage-related tasks with playbooks that are available for your OS or create your own playbooks.

Here are some of the functions that your Ansible playbooks can do:

- ▶ Create a file system.
- ▶ Remove a file system.
- ▶ Mount a file system.
- ▶ Unmount a file system.
- ▶ Create LVM volume groups.
- ▶ Remove LVM volume groups.
- ▶ Create logical volumes.
- ▶ Remove logical volumes.



## 6.1.2 Security and compliance

Each OS provides security technologies to combat vulnerabilities, protect data, and meet regulatory compliance. Depending on your location, industry, and the entities that you engage with, you might have different data protection regulations that you must adhere to.

Here are some examples of these regulations and standards:

- ▶ Department of Defense (DoD)
- ▶ General Data Protection Regulation (GDPR)
- ▶ Payment Card Industry Data Security Standard (PCI DSS)
- ▶ Defense Information Systems Agency (DISA) Security Technical Implementation Guide (STIG).
- ▶ Criminal Justice Information Services (CJIS) security policy.
- ▶ Commercial cloud services (C2Ss).
- ▶ Center for Internet Security (CIS)
- ▶ Health Insurance Portability and Accountability Act (HIPAA).
- ▶ NIST 800-171.
- ▶ Operating System Protection Profile (OSPP) 4.2.
- ▶ Red Hat Corporate Profile for Certified Cloud Providers.

Organizations that you work with might require companies in their supply chain to prove compliance by using an independent, third-party validation exercise. Ansible Automation Platform can be an optimal solution for an organization to automate regulatory compliance, security configuration, and remediation across systems and within containers. An organization can use existing playbook roles that are available in a community repository or they can develop their own playbooks and roles to meet their specific business requirements.

There are two roles that must be considered when you design your security playbooks. These roles can run in separate playbooks or they can be combined into a single playbook.

- ▶ Scanning playbook roles: This role scans systems based on the requirements that are set by the business. It also generates a report file that contains a list of the system updates that are required in your systems, and proof of compliance.
- ▶ Remediation playbook roles: This role applies to the appropriate system setting and the required changes to the systems based on business requirements or industry or governmental requirements for each OS.

## 6.1.3 Patches or upgrades

To keep your system up to date, you must complete the following tasks:

- ▶ Plan and configure how and when security updates are installed.
- ▶ Apply changes that are introduced by newly updated packages or file sets.
- ▶ Track security advisories.

As security vulnerabilities are discovered, the affected software must be updated to limit any potential security risks. Keeping your system up to date requires a patch management solution to manage and install updates. Updates can patch issues that are discovered, improve the performance of existing features, or add features to software. Patches and patch management solutions for each of the supported OSs for IBM Power are described in the OS-related sections later in this chapter.

## 6.1.4 Configuration and tuning

Configuration management is a process for maintaining computer systems, servers, applications, network devices, and other IT components to help ensure that they operate correctly. It is a way to help ensure that a system performs as expected, even after changes in the environment occur over time. Configuration management can include these activities and help teams to accomplish the following tasks:

- ▶ Classify and manage systems by groups and subgroups.
- ▶ Centrally modify base configurations.
- ▶ Roll out new settings to all applicable systems.
- ▶ Automate system identification, patches, and updates.
- ▶ Identify outdated, poorly performing, and non-compliant configurations.
- ▶ Prioritize the necessary actions.
- ▶ Access and apply prescriptive remediation.

Due to the scale and complexity of most enterprise environments, IT teams now use automation to define and maintain their various systems. For more information, see the Red Hat documentation at [Configuration Management](#).

## 6.2 Day 2 operations in Linux servers

Managing and maintaining your Linux servers can be time-consuming and manpower-intensive as the number of server images continues to grow. An automation solution that includes Ansible can help you manage the growing workload and make your system administrators more efficient. Automation simplifies the management process and can help you eliminate human errors as your staff does many repetitive tasks to maintain your server configuration and manage the required security fixes.

### 6.2.1 Installing system roles for Ansible automation

You can use a role that is defined for Red Hat Enterprise Linux (RHEL) automation (`rhel-system-roles`) that is a collection of Ansible roles and modules that provide a stable and consistent configuration interface to automate and manage RHEL across multiple releases. The effort is based on development of the [Linux System Roles](#) upstream project. There are also SAP related system roles that are provided by the [SAP LinuxLab](#) upstream project. Using these roles is described in this [Red Hat document](#) and 4.7, “SAP automation” on page 222.

For more information about the Red Hat Enterprise Linux System Roles, see [RHEL System Roles Overview](#).

Install the `rhel-system-roles` package on the Ansible Controller node by using the following command:

```
yum install rhel-system-roles -y
```

**Note:** The blivet application programming interface (API) packages are also needed. This API is the Python interface that you can use to create scripts to use for administration. To install the blivet API, use the yum command. The packages are blivet-data and python3-blivet.

For more information, see the following resources:

- ▶ [How to install the blivet API](#)
- ▶ [How to use the blivet API](#)

## 6.2.2 Storage

This section shows some simple tasks as examples of things that you might want to automate in your storage environment on your Linux on IBM Power VMs.

### Creating a file system by using the RHEL System Role for storage

In this section, you create an Ansible playbook that you use to create a file system in your RHEL virtual machine (VM). To do this task, complete the following steps:

1. Create a playbook by using the RHEL System Role that is called `rhel-system-roles.storage`, which is shown in Example 6-1.

*Example 6-1 Creating the `create_lvm_filesystem_playbook1.yaml` playbook*

---

```
# mkdir storage

# cat create_lvm_filesystem_playbook1.yaml
---
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/mapper/360050768108201d83800000000008e08p1
          - /dev/mapper/360050768108201d83800000000008e08p2
        volumes:
          - name: mylv1
            size: 1 GiB
            fs_type: xfs
            mount_point: /opt/mount1
  roles:
    - rhel-system-roles.storage
```

---

The simple playbook that is shown in Example 6-1 does the following tasks:

- Defines a `myvg` volume group, which should contain the following disks:
  - `/dev/mapper/360050768108201d83800000000008e08p1`
  - `/dev/mapper/360050768108201d83800000000008e08p2`
- Defines a logical volume (`mylv1`).
- If the `myvg` volume group exists, the playbook adds the logical volume `mylv1` to the volume group.

- If the myvg volume group does not exist, the playbook creates it.
  - The playbook creates an xfs file system on the mylv1 logical volume and persistently mounts the file system at /opt/mount1.
2. Check the inventory file for the list of target systems and then run the playbook that was created in Example 6-1 on page 283. The inventory file that is used in this example is the hosts file. The process is shown in Example 6-2.

*Example 6-2 Running the create\_lvm\_filesystem\_playbook1.yaml playbook*

```
# pwd
/root/storage

# ls -la
total 52
-rw-r--r--. 1 root root 39216 Aug 27 15:30 ansible.cfg
-rw-r--r--. 1 root root 298 Aug 27 15:35 create_lvm_filesystem_playbook1.yaml
-rw-r--r--. 1 root root 16 Aug 27 15:30 hosts
-rw-r--r--. 1 root root 319 Aug 27 15:54 resize_lvm_filesystem_playbook1.yaml
-rw-r--r--. 1 root root 320 Aug 27 16:14 resize_lvm_filesystem_playbook2.yaml

# cat hosts
bs-rbk-lnx-1.power-iaas.cloud.ibm.com

# ansible-playbook create_lvm_filesystem_playbook1.yaml
TASK [rhel-system-roles.storage : Set the list of pools for test verification]
*****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Set the list of volumes for test
verification] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Remove obsolete mounts]
*****
skipping: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Tell systemd to refresh its view of
/etc/fstab] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Set up new/current mounts]
*****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com] => (item={'src':
'/dev/mapper/myvg-mylv1', 'path': '/opt/mount1', 'fstype': 'xfs', 'opts':
'defaults', 'dump': 0, 'passno': 0, 'state': 'mounted'})

TASK [rhel-system-roles.storage : Tell systemd to refresh its view of
/etc/fstab] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [rhel-system-roles.storage : Retrieve facts for the /etc/crypttab file]
*****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]
```

```
TASK [rhel-system-roles.storage : Manage /etc/crypttab to account for changes
you just made] *****
skipping: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]
```

```
TASK [rhel-system-roles.storage : Update facts]
*****
*****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]
```

```
PLAY RECAP
*****
*****
bs-rbk-lnx-1.power-iaas.cloud.ibm.com : ok=21   changed=3   unreachable=0
failed=0   skipped=11   rescued=0   ignored=0
```

3. Verify the storage configuration by using the commands that are shown in Example 6-3.

*Example 6-3 Verifying the storage configuration*

```
# vgs
VG   #PV #LV #SN Attr   VSize VFree
myvg 2   1   0 wz--n- 19.99g 18.99g

# lvs
LV   VG   Attr          LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
mylv1 myvg -wi-ao---- 1.00g

# grep mylv1 /etc/fstab
/dev/mapper/myvg-mylv1 /opt/mount1 xfs defaults 0 0
# df -h |grep '/opt/mount1'
/dev/mapper/myvg-mylv1                1014M   40M   975M   4%
/opt/mount1
```

As verified in Example 6-3, the playbook created a volume group and logical volume. It also created an xfs file system and persistently mounted it as the /opt/mount1 directory.

## Resizing an existing LVM file system by using the RHEL System Role for storage

In this section, you create an Ansible playbook that resizes an existing LVM-based file system. The first step is to extend the existing volume group that was created in Example 6-2 on page 284.

**Note:** Attempt only one change at a time. Do not extend the volume group while resizing the existing file system in a single playbook.

## Extending the existing volume group

To extend the existing volume group, complete the following steps:

1. Add a disk in the existing volume group to provide space to extend the file system, as shown in Example 6-4.

*Example 6-4 Creating the `resize_lvm_filesystem_playbook1.yaml` playbook*

---

```
# cat resize_lvm_filesystem_playbook1.yaml
---
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/mapper/360050768108201d83800000000008e08p1
          - /dev/mapper/360050768108201d83800000000008e08p2
          - /dev/mapper/360050768108201d83800000000008e08p2
        volumes:
          - name: mylv1
            size: 1 GiB
            fs_type: xfs
            mount_point: /opt/mount1
  roles:
    - rhel-system-roles.storage
```

---

2. Copy some files to the `/opt/mount1` mount point to validate that it is available, and then run the second playbook, as shown in Example 6-5.

*Example 6-5 Running the `resize_lvm_filesystem_playbook1.yaml` playbook*

---

```
# cp /etc/fstab /opt/mount1/
# cp /etc/hosts /opt/mount1/
# ls -l /opt/mount1/
total 8
-rw-r--r--. 1 root root 146 Aug 27 22:17 fstab
-rw-r--r--. 1 root root 225 Aug 27 22:17 hosts

# ansible-playbook resize_lvm_filesystem_playbook1.yaml
```

---

3. Verify the changed storage configuration, as shown in Example 6-6.

*Example 6-6 Verifying the new storage configuration*

---

```
# vgs
VG #PV #LV #SN Attr VSize VFree
myvg 3 1 0 wz--n- <29.99g <28.99g

# lvs
LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync
Convert
mylv1 myvg -wi-ao----- 1.00g

# ls -l /opt/mount1/
total 8
-rw-r--r--. 1 root root 146 Aug 27 22:17 fstab
-rw-r--r--. 1 root root 225 Aug 27 22:17 hosts
# cat /opt/mount1/hosts
```

```
127.0.0.1      localhost localhost.localdomain localhost4
localhost4.localdomain4
::1           localhost localhost.localdomain localhost6
localhost6.localdomain6
192.168.159.133 bs-rbk-lnx-1.power-iaas.cloud.ibm.combs-rbk-lnx-1
```

---

As verified in Example 6-6 on page 286, the playbook expanded the existing volume group, but the logical volume, file system and persistent mount point remain the same and the data in the file system is still accessible.

### **Resizing the existing file system**

Resize the existing file system by completing the following steps:

1. Create the playbook to resize the existing file system, as shown in Example 6-7.

#### *Example 6-7 Creating the `resize_lvm_filesystem_playbook2.yaml` playbook*

---

```
# cat resize_lvm_filesystem_playbook2.yaml
---
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/mapper/360050768108201d83800000000008e08p1
          - /dev/mapper/360050768108201d83800000000008e08p2
          - /dev/mapper/360050768108201d83800000000008e08p2
        volumes:
          - name: mylv1
            size: 10 GiB
            fs_type: xfs
            mount_point: /opt/mount1
    roles:
      - rhel-system-roles.storage
```

---

2. Run the second playbook by running the following command:

```
ansible-playbook resize_lvm_filesystem_playbook1.yaml
```

3. Verify the storage configuration by running the commands in Example 6-8.

#### *Example 6-8 Verifying the storage configuration*

---

```
# vgs
VG #PV #LV #SN Attr VSize VFree
myvg 3 1 0 wz--n- <29.99g <28.99g

# lvs
LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
mylv1 myvg -wi-ao---- 10.00g

# df -h |grep '/opt/mount1'
/dev/mapper/myvg-mylv1 10G 106M 9.9G 2% /opt/mount1

# cat /opt/mount1/hosts
127.0.0.1      localhost localhost.localdomain localhost4 localhost4.localdomain4
::1           localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.159.133 bs-rbk-lnx-1.power-iaas.cloud.ibm.combs-rbk-lnx-1
```

---

As verified in Example 6-8 on page 287, the playbook expanded the existing logical volume along with file system and the data is still accessible

Other storage options can be used by using Ansible playbooks, such as the following tasks:

- ▶ Remove a file system.
- ▶ Unmount a file system.
- ▶ Remove LVM volume groups.
- ▶ Remove logical volumes.

For more information about how to use Ansible for these functions, see [Introduction to RHEL system roles](#).

In addition, the readme file for the role has more details about how to use the `rhel-system-roles.storage` role. You can find it on your controller system at the following directory:

```
/usr/share/ansible/roles/rhel-system-roles.storage/README.md
```

## 6.2.3 Security and compliance

There are some selective security checklists from different industry standards, such as PCI-DSS, DoD, and CIS that are considered sample baselines for this demonstration. Example 6-9 shows the `rhel-hardening-scanning` directory in the project directory that has multiple subdirectories followed by a specific directory structure.

*Example 6-9 Listing the subdirectories of the `rhel-hardening-scanning` project directory*

---

```
# tree -d rhel-hardening-scanning/
rhel-hardening-scanning/
... roles
... rhel8hardening
.   ... defaults
.   ... files
.   .   ... pam.d
.   ... handlers
.   ... tasks
.   ... templates
... rhel8scanning
.   ... defaults
.   ... files
.   .   ... pam.d
.   ... tasks
.   ... templates
```

---

Here is a basic introduction to these subdirectories and a description of their purpose:

- ▶ The `rhel-hardening-scanning` subdirectory
  - The project directory, which has multiple subdirectories and contains the main playbooks that are linked back to the role or other playbooks in different subdirectories. For example:
    - The file `rhel-hardening-scanning/playbook-rhel8hardening.yml` to harden the system according to your sample baselines.
    - The file `rhel-hardening-scanning/playbook-rhel8scanning.yml` to scan the system according to your sample baselines and generate a report file.



► The roles subdirectory

A defined directory structure that you can use to develop reusable automation components by grouping and encapsulating related automation artifacts, such as configuration files, templates, tasks, and handlers. We omitted some directories the role does not use.

► The rhel-hardening-scanning subdirectory includes the following subdirectories:

defaults	Contains the default variables for the role and defines all the required variables.
handlers	Contains a list of tasks that run only when a change is made on a machine, and run only after all the tasks in a playbook are completed.
files	Contains all the files that the role deploys.
templates	Contains all the configuration template files that the role deploys.
tasks	Contains the list of tasks that the role runs. The main list of tasks is in main.yml.

**Note:** For more information about Ansible roles, see the [Playbook Guide](#). For help with developing a role, see [Developing an Ansible Role](#).

The list of files under the subdirectory of the rhel-hardening-scanning project directory is shown in Example 6-10.

*Example 6-10 Listing of files under the subdirectory of the rhel-hardening-scanning project directory*

```
# tree -f rhel-hardening-scanning/
rhel-hardening-scanning
... rhel-hardening-scanning/ansible.cfg
... rhel-hardening-scanning/hosts
... rhel-hardening-scanning/playbook-rhel8hardening.yml
... rhel-hardening-scanning/playbook-rhel8scanning.yml
... rhel-hardening-scanning/roles
... rhel-hardening-scanning/roles/rhel8hardening
.  ... rhel-hardening-scanning/roles/rhel8hardening/defaults
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/defaults/main.yml
.  ... rhel-hardening-scanning/roles/rhel8hardening/files
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/files/chrony.conf
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/files/pam.d
.  .  .  ...
rhel-hardening-scanning/roles/rhel8hardening/files/pam.d/password-auth
.  .  .  ... rhel-hardening-scanning/roles/rhel8hardening/files/pam.d/su
.  .  .  ...
rhel-hardening-scanning/roles/rhel8hardening/files/pam.d/system-auth
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/files/rsyslog.conf
.  ... rhel-hardening-scanning/roles/rhel8hardening/handlers
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/handlers/main.yml
.  ... rhel-hardening-scanning/roles/rhel8hardening/tasks
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/tasks/main.yml
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/tasks/prerequisite.yml
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/tasks/section_A.yml
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/tasks/section_B.yml
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/tasks/section_C.yml
.  .  ... rhel-hardening-scanning/roles/rhel8hardening/tasks/section_D.yml
```

```

. . . . . rhel-hardening-scanning/roles/rhel8hardening/tasks/section_E.yml
. . . . . rhel-hardening-scanning/roles/rhel8hardening/tasks/section_F.yml
. . . . . rhel-hardening-scanning/roles/rhel8hardening/tasks/section_G.yml
. . . . . rhel-hardening-scanning/roles/rhel8hardening/templates
. . . . . rhel-hardening-scanning/roles/rhel8hardening/templates/login.defs.j2
... rhel-hardening-scanning/roles/rhel8scanning
    ... rhel-hardening-scanning/roles/rhel8scanning/defaults
    . . . . . rhel-hardening-scanning/roles/rhel8scanning/defaults/main.yml
    ... rhel-hardening-scanning/roles/rhel8scanning/files
    . . . . . rhel-hardening-scanning/roles/rhel8scanning/files/pam.d
    . . . . .
rhel-hardening-scanning/roles/rhel8scanning/files/pam.d/password-auth
. . . . . rhel-hardening-scanning/roles/rhel8scanning/files/pam.d/su
. . . . .
rhel-hardening-scanning/roles/rhel8scanning/files/pam.d/system-auth
. . . . . rhel-hardening-scanning/roles/rhel8scanning/files/rsyslog.conf
... rhel-hardening-scanning/roles/rhel8scanning/tasks
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/main.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/postreport.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/prerequisite.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_A-report.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_B-report.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_C-report.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_D-report.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_E-report.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_F-report.yml
. . . . . rhel-hardening-scanning/roles/rhel8scanning/tasks/section_G-report.yml
... rhel-hardening-scanning/roles/rhel8scanning/templates
    ... rhel-hardening-scanning/roles/rhel8scanning/templates/report.html.j2

```

---

## Running rhel-hardening-scanning from Ansible Controller nodes

Target systems or managed nodes need some preparation to run the playbooks from the Ansible Controller Node. For more information about that process, see “Getting started with Linux management” on page 32.

Example 6-11 shows how to run one of the provided playbooks to scan a system from the Ansible Controller node by using the Ansible CLI.

*Example 6-11 Running rhel-hardening-scanning from the Ansible Controller node*

```

# cd rhel-hardening-scanning/

# ls -l
total 32
-rw-r--r--. 1 root  root  19971 Aug 26 12:00 ansible.cfg
-rw-r--r--. 1 root  root1031 Aug 28 21:25 hosts
-rwxrwxrwx. 1 mhaque mhaque  123 Aug 26 11:43 playbook-rhel8hardening.yml
-rwxrwxrwx. 1 mhaque mhaque  125 Aug 26 11:58 playbook-rhel8scanning.yml
drwxrwxr-x. 4 mhaque mhaque   49 Aug 26 11:42 roles

# cat hosts
135.90.72.133

# ansible-playbook playbook-rhel8scanning.yml -u root

```

---

## Running rhel-hardening-scanning from Ansible Automation Platform

This example uses the Ansible Automation Platform to perform scanning and hardening. The required resources (Credentials, Inventories, Projects and Job Templates, and Workflow Job Templates) were created in the Ansible Automation Platform.

Figure 6-1 shows the window for Job Templates and Workflow Job Templates configuration.

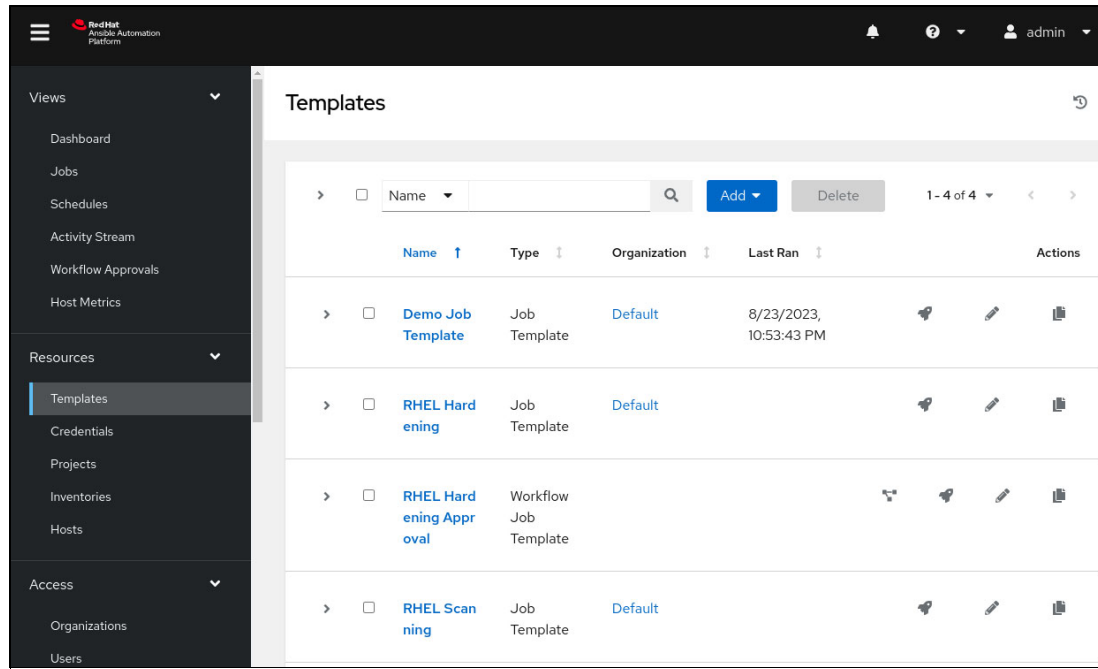


Figure 6-1 Job Templates and Workflow Job Templates

One benefit of the Ansible Automation Platform (beyond the GUI interface that is provided) is the additional management functions, such as requiring approval before starting any sensitive playbooks execution that might change system settings.

Figure 6-2 shows defining a Workflow Job Template that is configured to require approval.

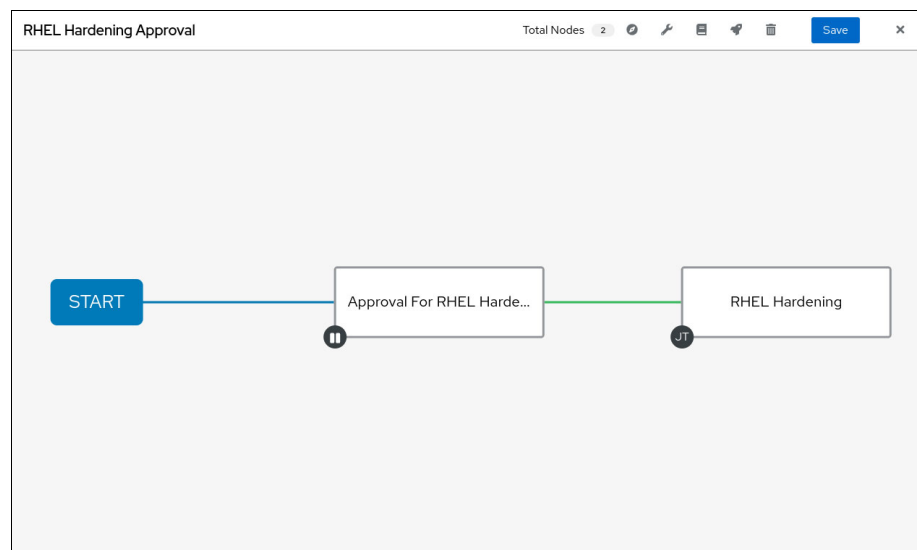


Figure 6-2 Workflow Job Templates configuration with approval

## Sample scanning report

Scanning provides a simple HTML template-based report file that is generated in the managed nodes (target systems) or in the Ansible Controller node. Figure 6-3 shows an example output from the upper part of the report file.

General Information			
Project Name		IBM RedBooks	
Computer Name		br-616-lex-1.power-lan.cloud.ibm.com	
MAC Address		fa:8b:56:66:14:20	
IP Address		192.168.159.133	
Subnet Mask		255.255.255.248	
Default Gateway		192.168.159.129	
RHEL Version		8.6	
Total Tasks		35	
Non-compliant		16	
NA		0	
Execution Date		06-26-21_11:02	
Non-compliant Summary			
Section A1	Create separate partition for tmp	FAILED	***Please Create Separate Partition Manually***
Section A2	Set noexec option for tmp partition	FAILED	***Only Valid when tmp is separated***
Section A3	Set noaudit option for tmp partition	FAILED	***Only Valid when tmp is separated***
Section A4	Set noexec option for tmp partition	FAILED	***Only Valid when tmp is separated***
Section A5	Create separate Partition for /var	FAILED	***Please Create Separate Partition Manually***
Section C2	Disable Send Packet Redirects	FAILED	
Section C3	Disable Source Routed Packet Acceptance	FAILED	
Section C4	Disable ICMP Redirect Acceptance	FAILED	
Section C5	Disable Secure ICMP Redirects Acceptance	FAILED	
Section D3	Create and Set Permissions on rsyslog Log Files	FAILED	
Section D4	Configure rsyslog to Send Logs to a Remote Log Host	FAILED	Exception By Default
Section D5	Keep All Auditing Information	FAILED	
Section E3	Set User Group Owner and Permission	FAILED	
Section E4	Restrict all Daemons	FAILED	
Section E5	Restrict access to Authorized Users	FAILED	
Section F4	Disable System Accounts	FAILED	
NA	Tasks	PASSED/FAILED	Remarks
Section A: Install Updates, Patches and Additional Security Software			

Figure 6-3 Top of the report files

Figure 6-4 shows an example of the lower part of the report file.

Section B5	Remove NIS Client	PASSED	
Section C: Network Configuration and Firewall			
Section C1	Disable IP Forwarding	PASSED	
Section C2	Disable Send Packet Redirects	FAILED	
Section C3	Disable Source Routed Packet Acceptance	FAILED	
Section C4	Disable ICMP Redirect Acceptance	FAILED	
Section C5	Disable Secure ICMP Redirect Acceptance	FAILED	
Section D: Logging and Auditing			
Section D1	Install the rsyslog package	PASSED	
Section D2	Activate the rsyslog Service	PASSED	
Section D3	Create and Set Permissions on rsyslog Log Files	FAILED	
Section D4	Configure rsyslog to Send Logs to a Remote Log Host	FAILED	Exception By Default
Section D5	Keep All Auditing Information	FAILED	
Section E: System Access, Authentication and Authorization			
Section E1	Enable anacron Daemon	PASSED	
Section E2	Enable cron Daemon	PASSED	
Section E3	Set User Group Owner and Permission	FAILED	
Section E4	Restrict all Daemons	FAILED	
Section E5	Restrict access to Authorized Users	FAILED	
Section F: User Accounts and Environment			
Section F1	Set Password Expiration Days	PASSED	
Section F2	Set Password Change Minimum Number of Days	PASSED	
Section F3	Set Password Expiring Warning Days	PASSED	
Section F4	Disable System Accounts	FAILED	
Section F5	Set Default Group for root Account	PASSED	
Section G: Review User and Group Settings			
Section G1	Ensure Password Fields are Not Empty	PASSED	
Section G2	Verify No Legacy "*" Entries Exist in	PASSED	
Section G3	Verify No UID 0 Accounts Exist Other Than root	PASSED	
Section G4	Ensure root PATH Integrity	PASSED	
Section G5	Check Permissions on User Home Directories	PASSED	
RHEL: Hardening Report generated by Red Hat Ansible Automation Tools.			

Figure 6-4 Bottom of the report files

The sample playbooks for security and compliance for Linux on IBM Power System are available at [GitHub](#).

## 6.2.4 Patches and upgrades

One of the most important aspects of system security is keeping systems up to date with patches. You might have hundreds or thousands of RHEL servers in their environments, and tracking patches on servers can be challenging. But, if critical patches are missed on systems, it might result in the systems being compromised, having unscheduled downtime, or other issues.

Red Hat Insights is a software as a service (SaaS) offering that is included with your RHEL subscription. It includes several capabilities to help with various aspects of management. The Patch capability can help customers understand which advisories are applicable in their environments, and can help automate the process of patching through Ansible playbooks.

For example, if a Red Hat Security Advisory were issued, you can go into the Insights Patch dashboard to see a list of systems in your environment that are impacted. With a few clicks from within the Patch dashboard, you can generate an Ansible Playbook that can automate the advisory installation.

Figure 6-5 shows the GUI window that is used to create the Ansible playbook.

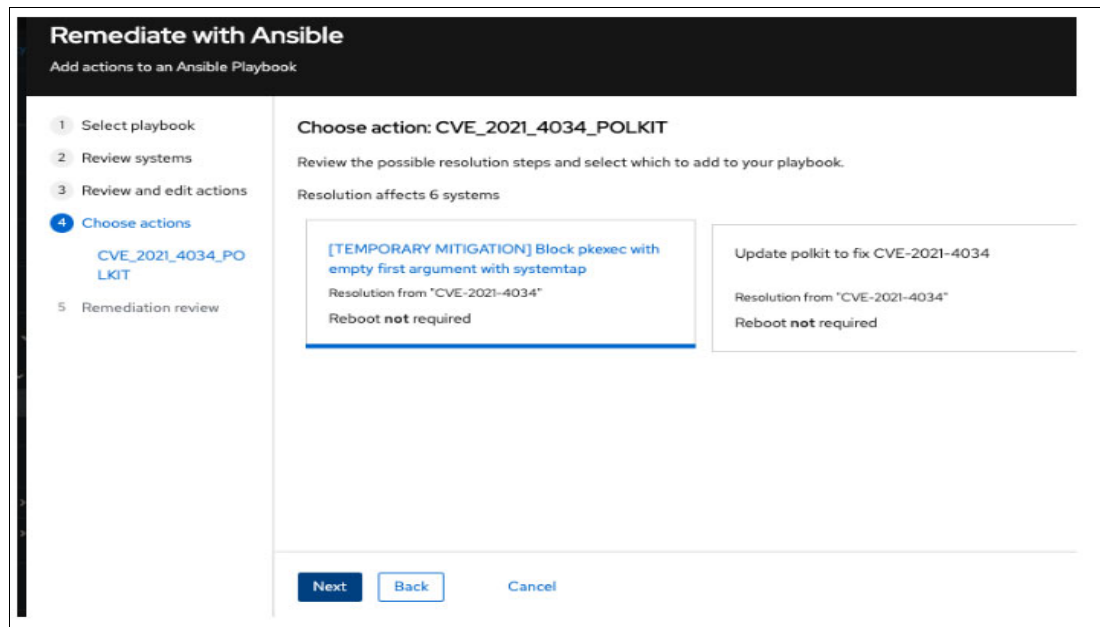


Figure 6-5 An example remediation playbook generation from Red Hat Insights<sup>1</sup>

If you have Red Hat Smart Management, you can use the Cloud Connector function to run the Ansible playbook from the Insights web interface. Smart Management, Satellite, and Cloud Connector are not required for use with Insights, and if you are in an environment without Red Hat Satellite, you can still use Insights Patch and generate Ansible playbooks that can be downloaded and manually run.

For more information about getting started with the Red Hat Insights patch capability and how to download Ansible playbooks, see this [Red Hat document](#).

<sup>1</sup> Source:

[https://docs.redhat.com/en/documentation/red\\_hat\\_insights/1-latest/html/red\\_hat\\_insights\\_remediation\\_s\\_guide/creating-managing-playbooks\\_red-hat-insights-remediation-guide](https://docs.redhat.com/en/documentation/red_hat_insights/1-latest/html/red_hat_insights_remediation_s_guide/creating-managing-playbooks_red-hat-insights-remediation-guide)

## Prerequisites

One of the following two options must be in place to pull fixes and upgrades from the Red Hat repositories.

1. Use Red Hat Satellite with the Red Hat Insights patch capability to enable and manage a standard operating environment for the patches or fixes repository. For more information, see this [Red Hat patch management document](#).
2. Alternatively, you can provide an individual RHEL subscription and connect with Red Hat Insights.

## Example Ansible playbook for RHEL OS patching

We have a Power System RHEL logical partition (LPAR) (bs-rbk-ln x-1) that is registered with a Red Hat Subscription and connected with Red Hat Insights. Example 6-12 shows the command to validate that the LPAR is registered with Red Hat Insights.

*Example 6-12 Verifying Red Hat Insights registration of the system*

```
[root@bs-rbk-lnx-1 ~]# insights-client --status
System is registered locally via .registered file. Registered at
2023-08-23T10:16:59.487964
Insights API confirms registration.
```

We downloaded an Ansible playbook from the [Red Hat Insights web console](#) (for advisory patches), as shown in Figure 6-6.

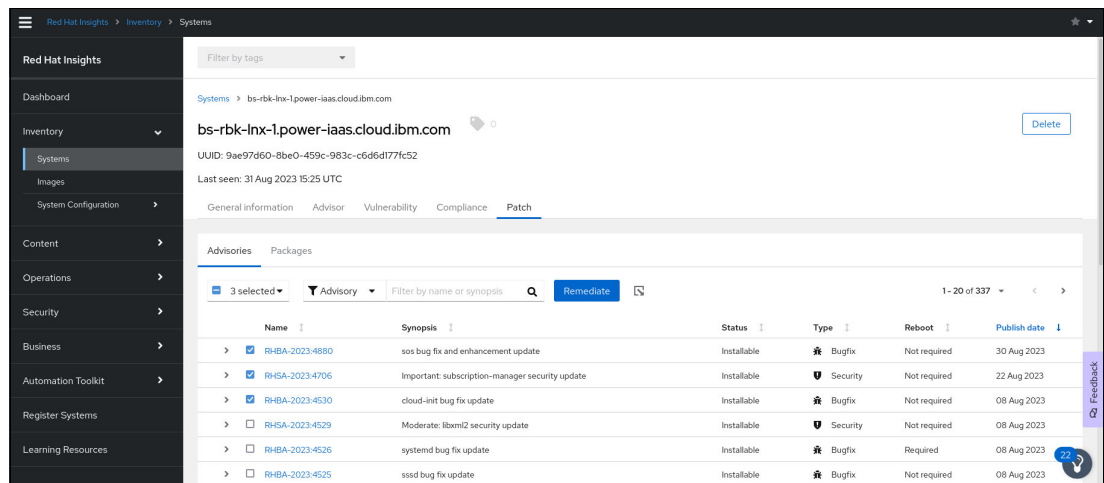


Figure 6-6 Creating a playbook (remediations) to apply patches from Red Hat Insights

After customizing some of the variables, the playbook that is shown in Example 6-13 runs.

*Example 6-13 Verifying and running the OS patch playbook*

```
# cat os-patch-playbook-check.yml
---
# Upgrade the following packages:
# - Apply RHBA-2023:4530
# - Apply RHBA-2023:4880
# - Apply RHSA-2023:4706
- name: update packages
  hosts: "bs-rbk-lnx-1.power-iaas.cloud.ibm.com"
  vars:
```

```

        insights_issues: "--advisory RHBA-2023:4530 --advisory
RHBA-2023:4880 --advisory RHSA-2023:4706"
requires_reboot: "true"
  become: true
  tasks:
- name: check for update
  shell: "{{ ansible_facts['pkg_mgr'] }}" check-update -q {{ insights_issues |
regex_search('(--advisory
((FEDORA-EPEL-[\w-]+)|(RH[SBE]A-20[\d]{2}:[\d]{4,5}))\s*+)' ) }}"
  check_mode: no
  register: check_out
  failed_when: check_out.rc != 0 and check_out.rc != 100

- when: check_out.rc == 100
  name: upgrade package
  shell: "{{ ansible_facts['pkg_mgr'] }}" update -d 2 -y {{ insights_issues |
regex_search('(--advisory
((FEDORA-EPEL-[\w-]+)|(RH[SBE]A-20[\d]{2}:[\d]{4,5}))\s*+)' ) }}"

- when: check_out.rc == 100
  name: set reboot fact
  set_fact:
    insights_needs_reboot: "{{requires_reboot}}"

# Restarts a system if any of the preceding plays sets the 'insights_needs_reboot'
variable to true.
# The variable can be overridden to suppress this behavior.
- name: Reboot system (if applicable)
  hosts: "bs-rbk-lnx-1.power-iaas.cloud.ibm.com"
  become: true
  gather_facts: false
  vars:
  tasks:
- when:
  - insights_needs_reboot is defined
  - insights_needs_reboot
  block:
  - name: Reboot system
    shell: sleep 2 && shutdown -r now "Ansible triggered reboot"
    async: 1
    poll: 0
    ignore_errors: true

  - name: Wait for system to boot up
    local_action:
      module: wait_for
      host: "{{ hostvars[inventory_hostname]['ansible_host'] |
default(hostvars[inventory_hostname]['ansible_ssh_host'], true) |
default(inventory_hostname, true) }}"
      port: "{{ hostvars[inventory_hostname]['ansible_port'] |
default(hostvars[inventory_hostname]['ansible_ssh_port'], true) | default('22',
true) }}"
      delay: 15
      search_regex: OpenSSH
      timeout: 300

```

become: false

```
# ansible-playbook os-patch-playbook-check.yml -u root
```

**Note:** Using customized yum or dnf commands help create a customized list of rpms for RHEL patching when an RHEL minor version upgrade is not supported by a running application or is restricted by the application.

Here are some example yum commands:

- ▶ yum --bugpatch check-update
- ▶ yum --security check-update
- ▶ yum --advisory check-update
- ▶ yum --secseverity=Important check-update
- ▶ yum --sec-severity=Critical check-update
- ▶ yum check-update --cve CVE-2008-0947
- ▶ yum check-update --cve CVE-2008-0947
- ▶ yum check-update --bz 1305903
- ▶ yum updateinfo list
- ▶ yum updateinfo list security all
- ▶ yum updateinfo list bugpatch all
- ▶ yum info-sec

For more information, see the [YUM command cheat sheet](#).

Figure 6-7 shows the results from running the playbook.

```
[mhaque@munshi-lab ibm_redbooks]$ ansible-playbook os-patch-playbook-check.yml -u root
PLAY [update packages] *****
TASK [Gathering Facts] *****
The authenticity of host 'bs-rbk-lnx-1.power-iaas.cloud.ibm.com (135.90.72.133)' can't be established.
ECDSA key fingerprint is SHA256:d0C0DaaNGAzidwS+XZzU8Hu7cNwoJlRfEt5DQLb2es.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [check for update] *****
[WARNING]: Consider using the dnf module rather than running 'dnf'. If you need to use command because dnf is insufficient you can add
'warn: false' to this command task or set 'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [upgrade package] *****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [set reboot fact] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

PLAY [Reboot system (if applicable)] *****

TASK [Reboot system] *****
changed: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com]

TASK [Wait for system to boot up] *****
ok: [bs-rbk-lnx-1.power-iaas.cloud.ibm.com -> localhost]

PLAY RECAP *****
bs-rbk-lnx-1.power-iaas.cloud.ibm.com : ok=6  changed=3  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

Figure 6-7 Output of the OS patch playbook

The system was patched with selective advisory rpms and successfully restarted because some of the rpms required a system restart.



## 6.2.5 Configuration tuning

You learned about the RHEL System Roles in 6.2, “Day 2 operations in Linux servers” on page 282. These roles that can help you configure system services in your system. For example, you can configure the following functions:

- ▶ Secure Shell (SSH) server and client
- ▶ Firewall and SELinux
- ▶ Time synchronization
- ▶ Networking
- ▶ Postpatch (mail transfer agent)
- ▶ High availability (HA) clustering
- ▶ Kernel settings

### Dynamic configuration files templates

With Ansible templates, you can create files dynamically by interpolating variables or by using logical expressions, such as conditionals and loops. It is useful to define configuration files that adapt to different contexts without having to manage extra files.

The Ansible template engine uses the [Jinja2](#) template language, a template language for the Python ecosystem. With Jinja2, you can interpolate variables and expressions with regular text by using special characters such as `{` and `{%`. By doing so, you can keep most of the configuration file as regular text and inject logic only when necessary, making it simpler to create, understand, and maintain template files.

**Note:** For more information, see [How to create dynamic configuration files using Ansible templates](#).

Jinja2 templates are files that use variables to include static values and dynamic values. One powerful thing about a template is that you can have a basic data file but use variables to generate values dynamically based on the destination host. Ansible processes templates by using Jinja2.

Example 6-14 shows how to create a file that is called `index.html.j2` for an Apache web server.

*Example 6-14 Sample index.html.j2 jinja2 templates file*

---

```
# cat index.html.j2
Welcome to {{ ansible_hostname }}
-The ipv4 address is {{ ansible_default_ipv4['address'] }}
-The current memory usage is {{ ansible_memory_mb['real']['used'] }}mb out of {{
ansible_memory_mb['real']['total'] }}mb
-The {{ ansible_devices | first }} block device has the following partitions:
-{{ ansible_devices['vdb']['partitions'] | join('\n -')}}
```

---

Example 6-15 shows how to use the template that was created in Example 6-14 on page 297 in an Ansible playbook.

*Example 6-15 Using the index.html.j2 jinja2 templates file in a playbook*

---

```
***** some output removed *****
# Push index Config Template
  - name: push index.html template
    template:
      src: index.html.j2
      dest: /var/www/html/index.html
***** some output removed *****
```

---

**Note:** For more information, see [How to manage Apache web servers using Jinja2 templates and filters](#).

**Example of a firewall configuration by using RHEL System Roles**

You can use Ansible to do system configuration by using the RHEL System Role.

Example 6-16 shows how to create a playbook file (~/opening-a-port.yml), which passes incoming HTTPS traffic to the local host.

*Example 6-16 Firewall to pass incoming HTTPS traffic to the local host*

---

```
# cat ~/opening-a-port.yml
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Allow incoming HTTPS traffic to the local host
  include_role:
    name: rhel-system-roles.firewall
vars:
  firewall:
    - port: 443/tcp
      service: http
      state: enabled
      runtime: true
      permanent: true
```

---

**Example of an SSHD configuration that uses RHEL System Roles**

Example 6-17 shows creating a playbook file (~/config-sshd.yml) to configure the sshd service to allow only root login (with a password) from a specific subnet.

*Example 6-17 Configuring the sshd service in the local host*

---

```
# cat ~/config-sshd.yml
---
- hosts: all
  tasks:
    - name: Configure sshd to prevent root and password login except from specific
      subnet
  include_role:
    name: rhel-system-roles.sshd
vars:
  sshd:
```

```
# root login and password login is enabled only from a particular subnet
PermitRootLogin: no
PasswordAuthentication: no
Match:
- Condition: "Address 192.0.2.0/24"
  PermitRootLogin: yes
  PasswordAuthentication: yes
```

---

### **Example of kernel parameter settings by using RHEL System Roles**

This section shows how to create a playbook file (`~/kernel-settings.yml`) that configures some kernel settings to provide better performance of the system. The playbook is defined in Example 6-18.

*Example 6-18 Configuring kernel settings in the local host*

---

```
# cat ~/kernel-settings.yml
---
- hosts: testingservers
  name: "Configure kernel settings"
  roles:
- rhel-system-roles.kernel_settings
  vars:
kernel_settings_sysctl:
  - name: fs.file-max
    value: 400000
  - name: kernel.threads-max
    value: 65536
kernel_settings_sysfs:
  - name: /sys/class/net/lo/mtu
    value: 65000
kernel_settings_transparent_hugepages: madvise
```

---

## **6.3 Day 2 operations in AIX environments**

This section describes some of the most common tasks on your AIX systems that you can automate with Ansible. This list of tasks is not complete. These tasks are examples of how you can automate your environments. Use and definitely amend them as needed before using them in your environment.

### **6.3.1 Storage**

Storage management is a common task that many administrators face on a daily basis. You can automate storage management tasks with Ansible and give your playbooks to your operating team to win more time for important tasks.

## Getting information about existing devices

Before you start working with storage devices, you must understand what you have on the system and how you can access the information.

When an Ansible playbook starts, its first task is to collect some information about the running system, that is, gathering facts. The information that Ansible collect is available through the variable `<ansible_facts>`. This variable has some parts regarding storage configuration. First, there is a list that is called `ansible_facts.devices` with the information about all devices on the system, including storage devices. You can also find a list of all mounted file systems in `ansible_facts.mounts`. Your volume groups are listed in `ansible_facts.vgs`.

One of the first problems almost every young administrator has on AIX is the absence of `df -h` (human readable output of all mounted file systems). You can solve this problem by using Ansible, which stores information about mounted file systems in `ansible_facts.mounts`. You can print this information, as shown in Example 6-19.

*Example 6-19 Printing information about mounted file systems in a human-readable format*

---

```
---
- name: print information about mounted file systems in human readable format
  host: all
  gather_facts: true

  tasks:
  - name: print mounted filesystems
    ansible.builtin.debug:
      msg: "{{ item.device }} {{ item.size_total | ansible.builtin.human_readable
    }} {{ item.size_available | ansible.builtin.human_readable }} {{ item.mount }}"
    loop: "{{ ansible_facts.mounts | sort(attribute='mount') }}"
    loop_control:
      label: "{{ item.device }}"
```

---

You can run Ansible on several hosts in parallel by running the distributed `df -h` command.

When working with storage, you should understand how you get your disks from a Virtual I/O Server (VIOS). You can find this information in the `ansible_facts.devices` tree. If you find disks of the type “Virtual SCSI Disk Drive”, you are using VSCSI. If you find disks of type “MPIO IBM 2145 FC Disk” or similar, you are using NPIV.

Example 6-20 shows using Ansible facts to differentiate NPIV and VSCSI disks in a storage area network (SAN) attached IBM DS8000®.

*Example 6-20 Finding different types of disks on AIX*

---

```
---
- name: find different types of disks on AIX
  hosts: all
  gather_facts: true

  tasks:
  - name: find VSCSI disks
    ansible.builtin.set_fact:
      vscsi_disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
    vars:
      q: "[?value.type == 'Virtual SCSI Disk Drive'].{ name: key }"
  - name: find NPIV disks
```

```

    ansible.builtin.set_fact:
      npiv_disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
    vars:
      q: "[?contains(value.type, 'IBM 2145')].{ name: key }"
  - name: VSCSI disks on the system
    ansible.builtin.debug:
      var: vscsi_disks
  - name: NPIV disks on the system
    ansible.builtin.debug:
      var: npiv_disks

```

---

One of the common tasks in AIX system administration is to create a volume group on a new disk. How can you find out which disk is new? In Ansible, start `cfgmgr` to find new disks and then recollect the information about devices. The difference between the two fact sets is your new disk. Example 6-21 demonstrates this capability.

*Example 6-21 Finding new disks on AIX*

---

```

---
- name: find new disk
  hosts: all
  gather_facts: true

  tasks:
  - name: find all existing hdisks
    ansible.builtin.set_fact:
      existing_disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
    vars:
      q: "[?starts_with(key, 'hdisk')].{ name: key }"
  - name: search for new disks
    ansible.builtin.command:
      cmd: cfgmgr
      changed_when: false
  - name: renew facts
    ansible.builtin.gather_facts:
      ignore_errors: true
  - name: get new list of disks
    ansible.builtin.set_fact:
      disks_after_cfgmgr: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
    vars:
      q: "[?starts_with(key, 'hdisk')].{ name: key }"
  - name: get new disks
    ansible.builtin.set_fact:
      new_disks: "{{ disks_after_cfgmgr |
ansible.builtin.difference(existing_disks) }}"
  - name: print new disks
    ansible.builtin.debug:
      var: new_disks

```

---

The resulting output from running the playbook is shown in Example 6-22.

*Example 6-22 Output from the playbook*

---

```
# ansible-playbook -i localhost, -c local find-new-disk.yml

PLAY [find new disk]
*****
*****

TASK [Gathering Facts]
*****
*****

[WARNING]: Platform aix on host localhost is using the discovered Python
interpreter at /opt/freeware/bin/python3.9, but future installation of
another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.14/reference_appendices/interpreter_discovery.html for more information.
ok: [localhost]

TASK [find hdisks]
*****
*****

ok: [localhost]

TASK [search for new disks]
*****
*****

ok: [localhost]

TASK [renew facts]
*****
*****

ok: [localhost]

TASK [get new list of disks]
*****
*****

ok: [localhost]

TASK [get new disks]
*****
*****

ok: [localhost]

TASK [print new disks]
*****
*****

ok: [localhost] => {
  "new_disks": [
    {
      "name": "hdisk6"
    }
  ]
}
```

## PLAY RECAP

```
*****  
*****  
localhost           : ok=7   changed=0   unreachable=0   failed=0   skipped=0  
rescued=0   ignored=0
```

---

After you find the disk, you might want to set attributes for it like `reserve_policy` or `hcheck_mode`. To do so, use the `ibm.power_aix.devices` module, as shown in Example 6-23.

### *Example 6-23 Setting device attributes for new disks*

---

```
- name: set attributes for new disks  
  ibm.power_aix.devices:  
    device: "{{ item.name }}"  
    attributes:  
      reserve_policy: "no_reserve"  
      hcheck_mode: "enabled"  
  loop: "{{ new_disks }}"  
  loop_control:  
    label: "{{ item.name }}"
```

---

## Working with volume groups

After you find the disks to work with, you might want to create a volume group on it. The module to work with volume groups is called `ibm.power_aix.lvg`.

Example 6-24 shows how to create the volume.

### *Example 6-24 Creating a volume group on AIX*

---

```
---  
- name: create volume group  
  hosts: all  
  gather_facts: false  
  
  tasks:  
    - name: create volume group  
      ibm.power_aix.lvg:  
        vg_name: datavg  
        pvs: hdisk6  
        vg_type: scalable  
        pp_size: 256  
        state: present
```

---

The created volume group is automatically activated and ready for further work, such as creating logical volumes or file systems.

If you use the variable `<new_disks>` in Example 6-22 on page 302 instead of inputting the name manually, you must build a string pointing to the variable, as shown in Example 6-25.

*Example 6-25 Creating a volume group by using a list of disks*

---

```
- name: create volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    pvs: "{{ new_disks | map(attribute='name') | join(' ') }}"
    vg_type: scalable
    pp_size: 256
    state: present
```

---

A similar method can be used to delete an unneeded volume group. You need only two attributes, the volume group's name and state, as shown in Example 6-26.

*Example 6-26 Deleting a volume group*

---

```
- name: delete volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    state: absent
```

---

The volume group is deleted even if it is open (varied on), but it cannot be deleted if there are allocations (LPARs) in it. In this case, you must collect LVM-related information, unmount all file systems that are in the volume group, remove all logical volumes, and then delete the volume group, as shown in Example 6-27.

*Example 6-27 Deleting a volume group with logical volumes in it*

---

```
---
- name: delete volume group
  hosts: all
  gather_facts: false
  vars:
    vgroup: datavg

  tasks:
    - name: gather LVM facts
      ibm.power_aix.lvm_facts:
    - name: "get logical volumes on {{ vgroup }}"
      ansible.builtin.set_fact:
        lvols: "{{ ansible_facts.LVM.LVs | dict2items |
community.general.json_query(q) }}"
      vars:
        q: "[?value.vg == '{{ vgroup }}'].{ name: key, mount: value.mount_point }"
    - name: unmount all filesystems
      ibm.power_aix.mount:
        state: umount
        mount_over_dir: "{{ item.mount }}"
        force: true
        loop: "{{ lvols }}"
    - name: remove all logical volumes
      ibm.power_aix.lvol:
        lv: "{{ item.name }}"
        state: absent
        loop: "{{ lvols }}"
```



```
- name: delete volume group
  ibm.power_aix.lvg:
    vg_name: "{{ vgroup }}"
    state: absent
```

---

Another common task is when you want to expand a volume group by adding disks to it or to shrink it by removing disks. It is the same procedure that is to create and delete a volume group. Example 6-28 shows adding a disk.

*Example 6-28 Adding a disk to a volume group*

---

```
- name: add disk to volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    pvs: hdisk6
    state: present
```

---

Example 6-29 shows removing a disk.

*Example 6-29 Removing a disk from a volume group*

---

```
- name: remove disk from volume group
  ibm.power_aix.lvg:
    vg_name: datavg
    pvs: hdisk6
    state: absent
```

---

The disk must be empty before removing it. If you need to move all logical volumes to another disk, use the `migratepv` command. At the time of writing, there is not a special module or role to free up a disk. However, you can use `ansible.builtin.command`, which passes any command to run as though you are running it on the CLI, as shown in Example 6-30.

*Example 6-30 Freeing up a disk by moving logical partitions to another disk*

---

```
- name: move all LPs to another volume
  ansible.builtin.command:
    cmd: "migratepv hdisk6 hdisk5"
```

---

## Working with logical volumes

You deleted logical volumes in “Working with volume groups” on page 303, but you did not create any logical volumes by using Ansible. To do so, use the logic that is shown in Example 6-31.

*Example 6-31 Creating a logical volume*

---

```
- name: create logical volume
  ibm.power_aix.lvol:
    vg: datavg
    lv: lv01
    lv_type: jfs2
    size: 1G
    state: present
```

---

Using this same logic with a logical volume name specified, but with the state specified as absent, the logical volume is deleted, as shown in Example 6-32.

*Example 6-32 Deleting a logical volume*

---

```
- name: delete logical volume
  ibm.power_aix.lvol:
    lv: lv01
    state: absent
```

---

Sometimes, you must change existing logical volumes. For example, a common failure during file system expansion occurs if the logical volume was sized too small. This failure is shown in Example 6-33.

*Example 6-33 The maximum allocation for a logical volume is too small*

---

```
# chfs -a size=+1G /lv01
0516-787 extendlv: Maximum allocation for logical volume lv01
is 128.
```

---

You can change the maximum allocation or any other value by using the `extra_opts` attribute for `ibm.power_aix.lvol`, as shown in Example 6-34.

*Example 6-34 Setting the maximum allocation for a logical volume by using extra\_opts*

---

```
- name: set maximum allocation for logical volume
  ibm.power_aix.lvol:
    vg: datavg
    lv: lv01
    size: 1G
    extra_opts: "-x 512"
    state: present
```

---

**Note:** In this case, you must specify the volume group name and the size of the logical volume even if it does not change. To see which options you can use with the `extra_opts` attribute, see the [chlv command](#).

## Working with file systems

To work with file systems, you can use different modules depending on the tasks that you want to automate. You can use `ibm.power_aix.filesystem`, `ibm.power_aix.mount`, or `ansible.builtin.mount`.

To create a file system when you have an existing logical volume, specify the logical volume name, as shown in Example 6-35.

*Example 6-35 Creating a file system on an existing logical volume*

---

```
- name: create filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv01
    device: lv01
    fs_type: jfs2
    auto_mount: true
    permissions: rw
    attributes: agblksize=4096,logname=INLINE
    state: present
```

---

If you do not have a prepared logical volume, you must specify a volume group name to use to create the logical volume and include the size of the logical volume that will be created, as shown in Example 6-36.

*Example 6-36 Creating a file system on a new logical volume*

---

```
- name: create filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv02
    vg: datavg
    fs_type: jfs2
    auto_mount: true
    permissions: rw
    attributes: agblksize=4096,logname=INLINE,size=1G
    state: present
```

---

After you create a file system, mount it as shown in Example 6-37.

*Example 6-37 Mounting the file system*

---

```
- name: mount filesystem
  ibm.power_aix.mount:
    mount_dir: /lv01
    state: mount
```

---

If you want to change the mount point of an existing file system, there is no special module for this task. You can unmount the file system, but you cannot change it by using the `ibm.power_aix.filesystem` module. To do this task, you must unmount the file system, run the `chfs` command, and then mount the file system on the new location.

*Example 6-38 Changing the mount point of a file system*

---

```
---
- name: change filesystem mount point
  hosts: all
  gather_facts: false

  tasks:
  - name: unmount filesystem
    ibm.power_aix.mount:
      state: umount
      mount_over_dir: /old_mount
      force: true
  - name: change mount point
    ansible.builtin.command:
      cmd: chfs -m /new_mount /old_mount
  - name: mount filesystem
    ibm.power_aix.mount:
      state: mount
      mount_dir: /new_mount
```

---

As an AIX administrator, sometimes you must change the size of a file system. One of the most significant advantages of AIX is that you can change your file system configuration dynamically. Example 6-39 shows how to expand an existing file system.

*Example 6-39 Expanding a file system*

---

```
- name: expand filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv02
    state: present
    attributes: size=+1G
```

---

Example 6-40 shows how to shrink a file system.

*Example 6-40 Shrinking a file system*

---

```
- name: shrink filesystem
  ibm.power_aix.filesystem:
    filesystem: /lv02
    state: present
    attributes: size=-1G
```

---

At the time of writing, there is a known bug in the `ibm.power_aix.filesystem` module. If there was an error during `chfs` execution and no parameters of the original file system were changed, you often still get a returned status of `OK` instead of `FAILED`. One of the reasons why `chfs` fails might be the maximum allocations value on the underlying logical volume, as described in “Working with logical volumes” on page 305. In this case, first find the new value for the maximum allocation, and then set it before changing the size of the file system, as shown in Example 6-41.

*Example 6-41 Expanding a file system and changing the underlying logical volume*

---

```
---
- name: expand filesystem
  hosts: all
  gather_facts: false
  vars:
    fs: /lv02
    size: 10G

  tasks:
    - name: get LVM facts
      ibm.power_aix.lvm_facts:
    - name: find logical volume for the filesystem
      ansible.builtin.set_fact:
        lvol: "{{ ansible_facts.LVM.LVs | dict2items |
community.general.json_query(q) | first }}"
      vars:
        q: "[?value.mount_point == '{{ fs }}'].{ lvname: key, vgname: value.vg,
mount: value.mount_point, lps: value.LPs }"
    - name: find pp size for the logical volume
      ansible.builtin.set_fact:
        ppsize: "{{ ansible_facts.LVM.VGs | dict2items |
community.general.json_query(q) | first | human_to_bytes | int }}"
      vars:
        q: "[?key == '{{ lvol.vgname }}'].value.pp_size"
    - name: recalculate new size in bytes
```

```

    set_fact:
      newsize: "{{ size | human_to_bytes | int }}"
- name: find new max lp alloc
  set_fact:
    maxlp: "{{ ((newsize | int) / (ppsize | int)) | round(0, 'ceil') | int }}"
- name: set max lp to logical volume
  ibm.power_aix.lvol:
    vg: "{{ lvol.vgname }}"
    lv: "{{ lvol.lvname }}"
    size: "{{ lvol.lps }}"
    extra_opts: "-x {{ maxlp }}"
    state: present
- name: expand filesystem
  ibm.power_aix.filesystem:
    filesystem: "{{ fs }}"
    state: present
    attributes: "size={{ size }}"

```

---

If you want to remove an existing file system, first unmount it, as shown in Example 6-42.

**Important:** AIX automatically deletes the underlying logical volume with all data on it if you remove a file system.

*Example 6-42 Removing a file system*

---

```

---
- name: delete filesystem
  hosts: all
  gather_facts: false

  tasks:
  - name: unmount filesystem
    ibm.power_aix.mount:
      mount_over_dir: /lv02
      state: umount
  - name: delete filesystem
    ibm.power_aix.filesystem:
      filesystem: /lv02
      state: absent

```

---

Because one of the common tasks in many environments is to add disks, create a volume group on those disks, and then create a file system on it that uses 100% of the disk. These tasks can be done by combining the code from the previous tasks to automate the whole workflow, as shown in Example 6-43.

*Example 6-43 Creating a file system on a disk that uses 100% of its space*

---

```

---
- name: create new filesystem on a new disk
  hosts: all
  gather_facts: true
  vars:
    vgname: vgora1
    lvname: lvora1
    fsname: /ora1

```

```

tasks:
- name: find hdisks
  ansible.builtin.set_fact:
    existing_disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
  vars:
    q: "[?starts_with(key, 'hdisk')].{ name: key }"
- name: search for new disks
  ansible.builtin.command:
    cmd: cfmgr
    changed_when: false
- name: renew facts
  ansible.builtin.gather_facts:
    ignore_errors: true
- name: get new list of disks
  ansible.builtin.set_fact:
    disks_after_cfmgr: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
  vars:
    q: "[?starts_with(key, 'hdisk')].{ name: key }"
- name: get new disks
  ansible.builtin.set_fact:
    new_disks: "{{ disks_after_cfmgr |
ansible.builtin.difference(existing_disks) }}"
- name: finish if no new disks are found
  ansible.builtin.meta: end_host
  when: new_disks | length == 0
- name: set attributes for new disks
  ibm.power_aix.devices:
    device: "{{ item.name }}"
    attributes:
      reserve_policy: "no_reserve"
      hcheck_mode: "enabled"
  loop: "{{ new_disks }}"
  loop_control:
    label: "{{ item.name }}"
- name: create volume group
  ibm.power_aix.lvg:
    vg_name: "{{ vgname }}"
    pvs: "{{ new_disks | map(attribute='name') | join(' ') }}"
    vg_type: scalable
    pp_size: 256
    state: present
- name: refresh LVM facts
  ibm.power_aix.lvm_facts:
    component: vg
- name: get size of the new volume group
  ansible.builtin.set_fact:
    vgs_size: "{{ ansible_facts.LVM.VGs | dict2items |
community.general.json_query(q) | first | int }}"
  vars:
    q: "[?key == '{{ vgname }}'].value.total_pps"
- name: create logical volume
  ibm.power_aix.lvol:

```

```

    vg: "{{ vname }}"
    lv: "{{ lvname }}"
    lv_type: jfs2
    size: "{{ vgsz }}"
    state: present
- name: create filesystem
  ibm.power_aix.filesystem:
    filesystem: "{{ fsname }}"
    device: "{{ lvname }}"
    fs_type: jfs2
    auto_mount: true
    permissions: rw
    attributes: agblksize=4096,logname=INLINE
    state: present
- name: mount filesystem
  ibm.power_aix.mount:
    mount_dir: "{{ fsname }}"
    state: mount
- name: set permissions on the mount point
  ansible.builtin.file:
    path: "{{ fsname }}"
    owner: root
    group: system
    mode: 0755
    state: directory

```

Now, you can compare how much time it takes to run all these commands manually and how much time it takes to run an Ansible playbook. Also consider that by changing the inventory files and variables, this playbook can be reused multiple times.

## 6.3.2 Security

Security is a broad topic with many nuances. It is impossible to describe the whole set of different configuration options that can be set in AIX to make it secure, but you will about some of them in this section. For more information about fixes, updates, and general configuration tuning, see 6.3.3, “Fixes” on page 317 and 6.4, “Day 2 operations in IBM i environments” on page 323.

### Managing users and groups

The first line in any security defense is to create user accounts for each user, set password rules for them, and lock and delete them if they are not needed anymore.

Start with creating users on AIX, which you can do by using the `ibm.power_aix.user` module, as shown in Example 6-44.

*Example 6-44 Creating a user on AIX*

---

```

- name: create user
  ibm.power_aix.user:
    name: user01
    state: present
    password:
      "{sha512}06$t/IwQ/bp8ygs5J3j$09w3kfyg/Zct2R8n63t7gYDnt1gi7z50CFa5wPxj.hfwEX4ALFUx
      8805n8MBAM5G1Ew7X4E7KG1ceNrp5XFW.."
    attributes:

```

```
id: 1001
shell: /usr/bin/bash
home: /home/user01
gecos: My fellow AIX admin
pgrp: staff
groups: staff,system,security
fsize: -1
```

---

All possible values for the attributes section can be found in [the chuser command](#). These values are standard AIX attributes that you usually use with the `mkuser` command.

During user creation, you can set the user's password, but it must be encrypted. It is copied one-to-one into `/etc/security/passwd`. If you want the user to change the password after the first login, add the attribute `change_passwd_on_login` to the task.

Unfortunately, the `ibm.power_aix.user` module is one of the few non-idempotent modules in the `ibm.power_aix` collection, which means that you should use different states if you want to create or to modify a user. You set the state to `present`, and Ansible creates a user if it does not exist or changes the specified attributes if it exists.

As an example, we create a simple password reset task for an AIX user. We generate a new password for the user and set the flag so that the user must change the password at the first login. At the end, we print the new password. This process is shown in Example 6-45.

*Example 6-45 Password reset*

---

```
---
- name: password reset
  gather_facts: false
  hosts: all
  vars:
    username: user01

  tasks:
    - name: generate random password
      ansible.builtin.set_fact:
        newpw: "{{ lookup('ansible.builtin.password', '/dev/null',
chars=['ascii_lowercase', 'ascii_uppercase', 'digits', '._-:'], length=8) }}"
    - name: encrypt password
      ansible.builtin.shell:
        cmd: "echo \"{{smd5}}$(echo \"{{ newpw }}\" | openssl passwd -aixmd5
-stdin)\\""
        changed_when: false
        register: newpw_enc
    - name: password reset
      ibm.power_aix.user:
        name: "{{ username }}"
        state: modify
        password: "{{ newpw_enc.stdout }}"
        change_passwd_on_login: true
    - name: show the generated password
      ansible.builtin.debug:
        msg: "The new password for {{ username }} is {{ newpw }}"
```

---

If you do not need the user anymore, you can set the state to `absent` and the user is deleted, as shown in Example 6-46 on page 313.



*Example 6-46 Deleting a user*

---

```
- name: delete user
  ibm.power_aix.user:
    name: user01
    state: absent
```

---

In a similar way, you can create and delete groups. Example 6-47 shows how to create a group.

*Example 6-47 Creating a group*

---

```
- name: create group
  ibm.power_aix.group:
    name: group01
    state: present
    group_attributes: id=1000,adms=user01
```

---

Example 6-48 shows deleting a group.

*Example 6-48 Deleting a group*

---

```
- name: delete group
  ibm.power_aix.group:
    name: group01
    state: absent
```

---

A common task on AIX systems is to add and to remove members from different groups, which you can implement by using Ansible and the `ibm.power_aix.group` module. Example 6-49 shows adding members to a group.

*Example 6-49 Adding members to an AIX group*

---

```
---
- name: add members to group
  gather_facts: false
  hosts: all
  vars:
    newmembers:
      - john
      - johann
      - ivan

  tasks:
    - name: add members to group
      ibm.power_aix.group:
        name: security
        state: modify
        user_list_action: add
        user_list_type: members
        users_list: "{{ newmembers }}"
```

---

Example 6-50 shows removing a user from a group.

*Example 6-50 Removing a user from a group*

---

```
---
- name: add members to group
  gather_facts: false
  hosts: all
  vars:
    rmmembers: ivan

  tasks:
  - name: add members to group
    ibm.power_aix.group:
      name: security
      state: modify
      user_list_action: remove
      user_list_type: members
      users_list: "{{ rmmembers }}"
```

---

**Note:** Note the inconsistency in the naming. The attributes `user_list_action` and `user_list_type` are singular (user without an s at the end), but the attribute `users_list` is plural (users with an s at the end).

## Managing the IP filter configuration

IP filter is an AIX tuning and configuration option that is not often used by AIX administrators, but more security departments require server-based firewalls that are enabled. With Ansible, it is simple to configure the IP filter on AIX.

To configure the IP filter without Ansible, you can use several smitty menus or learn the command and parameters to set the filters by using the CLI. With Ansible, you define a variable with your filter configuration and use one task to activate it, as shown in Example 6-51.

*Example 6-51 AIX IP filter configuration*

---

```
---
- name: configure AIX IP filter
  hosts: all
  gather_facts: false
  vars:
    aix_filter:
      - { action: permit, direction: outbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
        d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: all, description: 'allow outbound
        connections' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
        d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: icmp, interface: all, description:
        'allow incoming pings' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
        d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 22,
        description: 'ssh' }
      - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
        d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: all, d_opr: eq, d_port: 657,
        description: 'rnc' }
```

```

    - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
      d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 16191,
      description: 'caa' }
    - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
      d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 6181,
      description: 'caa' }
    - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
      d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 42112,
      description: 'caa' }
    - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
      d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 3901,
      description: 'nimsh' }
    - { action: permit, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
      d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: tcp, d_opr: eq, d_port: 3902,
      description: 'nimsh' }
    - { action: deny, direction: inbound, s_addr: 0.0.0.0, s_mask: 0.0.0.0,
      d_addr: 0.0.0.0, d_mask: 0.0.0.0, protocol: all, log: true, description: 'deny and
      log everything else' }
  tasks:
    - name: activate ip filter
      ibm.power_aix.mkfilt:
        ipv4:
          log: yes
          default: deny
          rules: "{{ aix_filter }}"

```

---

If you add a rule, you can add it into the list and run the playbook again. If you want to disable the IP filter, remove all filters, set default to permit, and then close the IPsec devices, as shown in Example 6-52.

*Example 6-52 Disabling the IP filter on AIX*

---

```

---
- name: disable AIX IP filter
  hosts: all
  gather_facts: false

  tasks:
    - name: Remove all user-defined and auto-generated filter rules
      ibm.power_aix.mkfilt:
        ipv4:
          default: permit
          force: yes
          rules:
            - action: remove
              id: all
    - name: stop IPsec devices
      ibm.power_aix.devices:
        device: "{{ item }}"
        state: defined
      with_items:
        - ipsec_v4
        - ipsec_v6

```

---

If you want to be sure that IP filter rules are loaded in the correct order, remove all rules before loading them again.

**Important:** Be careful when you change the IP filter configuration. Any mistake in the rules can lead to lost network connectivity.

## Managing security settings with aixpert

The aixpert tool is a versatile tool to help harden your system. With this command, you can check security settings and apply them. Ansible helps implement it in a playbook to make it a part of a larger process, such as server provisioning or hardening.

First, implement a highly secure configuration on AIX, as shown in Example 6-53.

### *Example 6-53 Applying a highly secure configuration*

---

```
- name: apply high secure configuration to AIX
  ibm.power_aix.aixpert:
    mode: apply
    level: high
```

---

Understand that it is aixpert that is doing the work this time, not Ansible. If you apply the configuration twice, Ansible runs aixpert twice. The tool does not change your configuration the second time and it might even be faster than the first run. It is aixpert that checks and applies security settings, not Ansible. To check whether the configuration is still intact, use check mode, as shown in Example 6-54.

### *Example 6-54 Checking the AIX security configuration*

---

```
- name: check applied settings
  ibm.power_aix.aixpert:
    mode: check
```

---

If the configuration changed, reapply it as shown in Example 6-55.

### *Example 6-55 Checking and reapplying AIX security settings if they changed*

---

```
- name: check applied settings
  ibm.power_aix.aixpert:
    mode: check
    ignore_errors: true
    register: aixpert_check
- name: reapply security settings
  ibm.power_aix.aixpert:
    mode: apply
    level: medium
  when: aixpert_check.rc == 1
```

---

If you want to restore your previous settings, you can do so because aixpert saves the old configuration, which you use to revert the changes, as shown in Example 6-56.

### *Example 6-56 Restoring security settings to their nonsecure variant*

---

```
- name: restore security settings
  ibm.power_aix.aixpert:
    mode: undo
```

---

### 6.3.3 Fixes

The topic of fixes, especially emergency or interim fixes, is a point of disagreement for many AIX administrators. Many administrators live according to the rule “if it works, don’t break it”. Unfortunately, what many administrators misunderstand is the first part of the rule: “if it works”.

If a security issue is found in some AIX component, then it does not work as it was designed. You don’t “break” it, you fix it. With Ansible, it is simple to patch these security issues. If your environment supports Live Update, fixing can be done while your systems are online and with no downtime.

#### Working with interim fixes

This section presents some examples about how you can use Ansible to manage interim fixes in your AIX environment. Some of the examples require direct access to the internet to work, but most corporate environments do not have any access to the internet or only through proxies.

If your systems do not have access to the internet, then you cannot automatically check and install fixes, and you must provide a method of checking and acquiring the appropriate fixes somewhere in your environment. You must have at least one server where you can download fixes that can then be distributed to the other systems.

If you do have access to the internet but only through a proxy, configure your proxy settings. Often, you can achieve this task by exporting the `https_proxy` variable in your environment, but sometimes the task requires more complex work depending on the specifics of your environment. Setting up proxy configurations can be automated with Ansible, but it is beyond the scope of this book, which is why we built our examples assuming that you have access to the internet.

**Note:** During our tests, we saw sometimes messages like `KeyError: <message>`. It seems that the FLRT service occasionally fails and delivers answers that are not understood by Ansible. Repeat the task, which should complete without any errors.

#### Checking for AIX fixes

Check whether there are fixes that are needed in your AIX installation. You need internet access, and you must install `wget` to download the fixes. The `wget` tool is an open-source utility that retrieves files by using the HTTP or FTP protocols. You can download it from the [AIX Toolbox](#) and install by using `dnf`. The module automatically installs the `flrtvc.ksh` script from the IBM website.

Example 6-57 shows how to generate a report on which fixes are available for your system.

*Example 6-57 Generating report about available security fixes for AIX*

---

```
- name: generate report about available fixes for the system
  ibm.power_aix.flrtvc:
    apar: sec
    verbose: true
    check_only: true
    register: flrtvc_out
- name: print the report
  ansible.builtin.debug:
    msg: "{{ flrtvc_out.meta['0.report'] | join('\n') }}"
```

---

If you want to see the report in a better format, export `ANSIBLE_STDOUT_CALLBACK=debug` or set it in the command when you call `ansible-playbook`, as shown in Example 6-58.

*Example 6-58 Setting debug before you run ansible-playbook*

---

```
# ANSIBLE_STDOUT_CALLBACK=debug ansible-playbook -i inventory fixes-report.yml
```

---

### **Installing fixes**

To automatically install the fixes, use the same command that you used to check for fixes, but remove `check_only`, as shown in Example 6-59.

*Example 6-59 Installing all the required security fixes on AIX*

---

```
- name: install security fixes
  ibm.power_aix.flrtvc:
    apar: sec
    verbose: true
```

---

Ansible downloads all the fixes into `/var/adm/ansible`. If there is not enough space in `rootvg`, Ansible automatically expands the file system. You can choose a different location for the temporary files by setting the attribute `path` to another directory.

### **Applying fixes to multiple servers**

Often, you have multiple similar systems that you want to manage as a group. You installed and tested the fixes in a non-production environment and now you want to apply those fixes to your production servers. Our example, which is shown in Example 6-60, assumes that the fixes that will be installed on the target production servers are downloaded and available in `/var/adm/fixes` on the Ansible Controller node.

*Example 6-60 Copying interim fixes from the Ansible Controller node to a remote AIX server and installing them*

---

```
---
- name: find, copy, and install security fixes
  hosts: all
  gather_facts: false
  vars:
    local_fixes_dir: /var/adm/fixes
    remote_fixes_dir: /var/tmp/fixes

  tasks:
    - name: find all fixes
      ansible.builtin.set_fact:
        fixes_list: "{{ fixes_list | default([]) + [ (item | basename) ] }}"
      with_fileglob:
        - "{{ local_fixes_dir }}/*.*epkg.Z"
    - name: create temporary directory for fixes on the target
      ansible.builtin.file:
        path: "{{ remote_fixes_dir }}"
        owner: root
        group: system
        mode: 0700
        state: directory
    - name: copy fixes to the target
      ansible.builtin.copy:
        src: "{{ local_fixes_dir }}/{{ item }}"
```

```

    dest: "{{ remote_fixes_dir }}/{{ item }}"
    owner: root
    group: system
    mode: 0600
  loop: "{{ fixes_list }}"
- name: install fixes
  ibm.power_aix.emgr:
    ipatch_package: "{{ remote_fixes_dir }}/{{ item }}"
    action: install
    from_epkg: true
    extend_fs: true
    force: true
  loop: "{{ fixes_list }}"
- name: remove fixes from the target
  ansible.builtin.file:
    path: "{{ remote_fixes_dir }}/{{ item }}"
    state: absent
  loop: "{{ fixes_list }}"

```

---

Sometimes, especially before performing an AIX update, you want to remove all fixes. You can do this task by using only two tasks in an Ansible playbook.

#### *Example 6-61 Removing all interim fixes from AIX*

---

```

---
- name: remove all installed interim fixes
  hosts: all
  gather_facts: false

  tasks:
  - name: find all installed fixes
    ibm.power_aix.emgr:
      action: list
      register: emgr
  - name: uninstall patch
    ibm.power_aix.emgr:
      action: remove
      ipatch_label: "{{ item.LABEL }}"
    loop: "{{ emgr.ipatch_details }}"
    loop_control:
      label: "{{ item.LABEL }}"

```

---

### **Installing service packs and updates from NIM lpp\_source resources**

An AIX environment has at least one network installation manager (NIM) server. The NIM server is a central focal point in the infrastructure that has the newest version of AIX, and it contains resources to manage NIM clients. The NIM administrator creates resources that can be used by NIM clients (AIX servers) to install new or update existing software.

Assume that you have an AIX server with AIX 7.3 and want to update it to AIX 7.3 TL1 SP2. The server is registered as an NIM client and has access to NIM resources. Example 6-62 shows an example of how you can update AIX by using the NIM `lpp_source` resource. It checks that the AIX server is registered while the NIM client validates that it does not have the update.

*Example 6-62 Updating the AIX server by using the NIM `lpp_source` from the NIM server*

---

```
---
- name: update AIX server by using NIM
  gather_facts: false
  hosts: nim
  vars:
    client: aix73
    aixver: 7300-01-02-2320
    reboot: false

  tasks:
    - name: check if client is defined
      ansible.builtin.command:
        cmd: lsnim {{ client }}
      changed_when: false
      register: registered
      ignore_errors: true
    - name: stop if the client is not registered
      meta: end_play
      when: registered.rc != 0
    - name: get client version
      ansible.builtin.command:
        cmd: /usr/lpp/bos.sysmgmt/nim/methods/c_rsh {{ client }} '( LC_ALL=C
/usr/bin/oslevel -s)'
      changed_when: false
      register: oslevel
    - name: stop if the client is already updated
      meta: end_play
      when: oslevel.stdout == aixver
    - name: update client
      ansible.builtin.command:
        cmd: nim -o cust -a lpp_source={{ aixver }}-lpp_source -a fixes=update_all
-a accept_licenses=yes {{ client }}
    - name: reboot client
      ibm.power_aix.reboot:
        when: reboot
```

---

You can update AIX from the NIM client too, as shown in Example 6-63.

*Example 6-63 Updating AIX by using the NIM `lpp_source` from the NIM client*

---

```
---
- name: update AIX server by using NIM
  gather_facts: false
  hosts: aix73
  vars:
    aixver: 7300-01-02-2320
    reboot: false
```



```

tasks:
- name: check if NIM client is configured
  ansible.builtin.command:
    cmd: nimclient -l master
  changed_when: false
  register: registered
  ignore_errors: true
- name: stop if the client is not registered
  meta: end_play
  when: registered.rc != 0
- name: get client version
  ansible.builtin.command:
    cmd: oslevel -s
  changed_when: false
  register: oslevel
- name: stop if the client is already updated
  meta: end_play
  when: oslevel.stdout == aixver
- name: update client
  ansible.builtin.command:
    cmd: nimclient -o cust -a lpp_source={{ aixver }}-lpp_source -a
fixes=update_all -a accept_licenses=yes
- name: reboot client
  ibm.power_aix.reboot:
  when: reboot

```

---

You can update any software that is packed in `lpp_source` on your NIM server.

### 6.3.4 Configuration tuning

There are many places in AIX that you can tune. In this section, you learn about several ways of tuning AIX.

Most of the AIX settings are stored in *stanza files*. They have sections, attributes, and values. You can change the values of attributes by using the `chsec` module, as shown in Example 6-64.

*Example 6-64 Changing the AIX settings in `/etc/security/login.cfg` and `/etc/secvars.cfg`*

---

```

- name: set settings
  ibm.power_aix.chsec:
    path: "{{ item.file }}"
    stanza: "{{ item.section }}"
    attrs: "{{ item.attrs }}"
  loop:
    - { file: "/etc/security/login.cfg", section: "usw", attrs: {
sologfulldate: "true", mkhomeatlogin: "true", pwd_algorithm: "ssha512" } }
    - { file: "/etc/secvars.cfg", section: "groups", attrs: {
domainlessgroups: "true" } }

```

---

Using the `chsec` module is not the only way to change AIX settings. You can use the standard `ansible.builtin.template` and `ansible.builtin.copy` modules to set AIX settings in the configuration files.

Security is not the only reason to automate AIX configuration. Another reason is maintaining performance baselines. An application vendor like Oracle or SAP can define some values that you must set up on AIX to achieve better performance. An AIX administrator usually does this task by using tunable commands like `vmo`, `no`, or `schedo`, or by setting device attributes. All these tasks are possible with Ansible.

Example 6-65 shows how you can set `reserve_policy` to `no_reserve` and `queue_depth` to 24 for each disk you find in the system.

*Example 6-65 Setting the hdisk attributes*

---

```
---
- name: set hdisk attributes
  hosts: all
  gather_facts: true

  tasks:
    - name: find hdisks
      ansible.builtin.set_fact:
        disks: "{{ ansible_facts.devices | dict2items |
community.general.json_query(q) }}"
      vars:
        q: "[?starts_with(key, 'hdisk')].key"
    - name: set hdisk attributes
      ibm.power_aix.devices:
        device: "{{ item }}"
        attributes:
          reserve_policy: no_reserve
          queue_depth: 24
        chtype: reset
      loop: "{{ disks }}"
```

---

Example 6-66 shows how you can set network tunables by using Ansible.

*Example 6-66 Setting network tunables with Ansible*

---

```
- name: set network options
  ibm.power_aix.tunables:
    action: modify
    component: no
    change_type: both
    tunable_params_with_value: "{{ no_tunables }}"
  vars:
    no_tunables: {
      tcp_recvspace: 262144,
      tcp_sendspace: 262144,
      udp_recvspace: 262144,
      udp_sendspace: 262144,
      rfc1323: 1,
      tcp_fastlo: 1,
      tcp_keepintvl: 60,
      ipforwarding: 0
    }
```

---

Similarly, you can other tunables like `vmo`, `schedo`, `ioo`, or `nfso`.

It is not possible to provide detailed descriptions of all options that are available or describe each use case. We tried to provide a little guidance and an initial impression of what can be done on AIX by using Ansible.

Ansible has a vibrant ecosystem. Every month, you see new features in Ansible collections and Ansible itself. If you cannot find some feature or module, send a report. Create an issue on GitHub or a topic on the IBM community site.

## 6.4 Day 2 operations in IBM i environments

In this segment, you learn about IBM i management through Ansible automation. This process encompasses a thorough investigation of essential elements, spanning storage, security, patch management, and configuration tuning. You learn about actionable solutions and industry best practices so that you can deftly choreograph IBM i environments for optimal performance and accuracy.

### 6.4.1 Storage

This section describes the context of storage management for IBM i. It aims to explore and demystify key storage-related tasks and configurations by harnessing the power of Ansible automation. By using Ansible functions, you can streamline and enhance the storage management experience for IBM i users and offer efficient solutions to common challenges.

Specifically, this section describes three integral facets of storage management:

1. Create or delete a storage volume by using `os_volume`: Describes using the `os_volume` module to facilitate the creation and deletion of storage volumes. This module is a versatile tool for managing storage resources with precision. It helps ensure that the storage landscape aligns with the dynamic requirements of IBM i.
2. Attach a storage volume to `os_server_volume`: The `os_server_volume` module is used to attach storage volumes to IBM i servers. You learn how Ansible simplifies and accelerates the provisioning of storage resources to enhance the scalability and adaptability of IBM i environments.
3. Configure volumes to independent auxiliary storage pool (IASP): IASPs can be used to help configure volumes within this context. With IASPs, you can create a storage infrastructure.

#### **Enhancing IBM i storage management with the `os_volume` module**

In IBM i storage management, the `os_volume` module enables the creation and removal of cinder block storage volumes. Use this tool to provision resources according to your evolving workload needs. Users can define whether to create or remove volumes to align storage operations with operational demands.

The `os_volume` module interacts seamlessly with designated clouds to improve operations by providing default authentication values. Users can specify target cloud environments to help ensure secure communication between Ansible and the cloud.

With this module, you have granular control over volume size (gigabytes) to accommodate precise resource allocation for IBM i. Volume naming enhances organization, and the module supports a volume type specification to tailor resources for various workloads.

Incorporating the `os_volume` module into Ansible simplifies storage management, which promotes efficient provisioning and configuration and exemplifies the dynamic storage landscape that is required by IBM i environments.

Example 6-67 shows a playbook for volume creation.

*Example 6-67 Sample playbook to create a volume on IBM i*

---

```
---
name: Create New Volume
os_volume:
  state: present
  cloud: '{{ cloud_name }}'
  size: 150
  display_name: "{{ volume_name }}"
  volume_type: '70866ebf-0db5-4b12-86ed-0e838d593458'
register: volume_info
...
```

---

## Attaching storage volumes to IBM i VMs by using the `os_server_volume` module

The `os_server_volume` module assumes a crucial role to facilitate the attachment of storage volumes to compute hosts. This module refines the process of linking volumes to IBM i VMs.

With its core function, `os_server_volume` presents administrators with the option to define the state of the resource, that is, whether it can be present or absent. By accommodating named clouds, this module permits precise targeting of cloud environments for the operation. Default authentication values simplify setup and bolster secure communication between Ansible and the cloud.

The module's efficacy is its capacity to associate volumes with specific IBM i VMs. By providing the name of the target VM and the volume, administrators can quickly attach storage resources to enable the VM to access the necessary data. This module exemplifies the synergy between the Ansible automation capabilities and the storage demands of IBM i.

Example 6-68 presents a sample playbook to attach a volume to an IBM i VM.

*Example 6-68 Sample playbook to attach a volume to an IBM i VM*

---

```
---
name: Attach Volume to IBM i VM
os_server_volume:
  state: present
  cloud: '{{ cloud_name }}'
  server: "{{ vm }}"
  volume: "{{ volume_info.id }}"
...
```

---

**Note:** For the effective utilization of the `os_server_volume` and `os_volume` modules, a prerequisite is the presence of IBM PowerVC as the orchestrator for your IBM i VMs. This integration emphasizes the significance of IBM PowerVC in bolstering the flexibility and resilience of your IBM i infrastructure.

## Configuring IASP volumes with Ansible for IBM i

In IBM i storage management, configuring volumes within an IASP is relevant for environments that uses IBM PowerHA, among others. The Ansible dynamic capabilities enhance this process by integrating storage resources into the IASP structure.

The `os_iasp_volume` role plays a crucial role in this configuration by efficiently orchestrating volume integration. The playbook first checks for the IASP's existence and creates it if needed, which shows Ansible's adaptability.

IASPs offer distinct benefits by enabling isolated disk unit management. The playbook demonstrates Ansible integration with `os_iasp_volume` to effortlessly configure non-configured disks in the IASP. This integration optimizes storage alignment to bolster overall efficiency.

The playbook highlights Ansible excellence in configuring IASP volumes. Administrators can expertly manage storage to help ensure a resilient landscape and meet evolving IBM i demands.

Example 6-69 presents a sample setup of IASP volumes.

*Example 6-69 Sample playbook to set up IASP volumes*

---

```
---
- name: Initialize IBM i VM
  hosts: new_vm
  roles:
    - role: vm_iasp_configure
  vars:
    iasp_info:
      - {'iasp_name': 'demoiasp1', 'iasp_capacity': 0.5}
      - {'iasp_name': 'demoiasp2', 'iasp_capacity': 1}
...

```

---

## 6.4.2 Security and compliance

Security in IBM i environments is a multifaceted aspect that plays a key role in maintaining the integrity and confidentiality of critical data and operations. At its core, the system offers five distinct security levels that are denoted by the QSECURITY system value, which ranges 10 - 50. Administrators can use these levels to tailor security measures to their specific needs. IBM i provides a comprehensive set of system values that administrators can use to define system-wide security settings while also facilitating customization to meet diverse requirements across IBM Power servers.

Digital signing emerges as a critical practice for helping ensure the authenticity and integrity of software objects. This situation becomes especially pertinent when objects traverse the internet or are on media that could be susceptible to unauthorized modifications. The usage of digital signatures, which are managed through mechanisms like the Verify Object Restore (QVFOBJRST) system value and the Check Manager tool, aids in detecting any unauthorized alterations.

Single sign-on (SSO) amplifies user convenience by granting access to multiple systems with a single set of credentials. IBM facilitates SSO through the Network Authentication Service (NAS) and Enterprise Identity Mapping (EIM), which both use the Kerberos protocol for user authentication. User profiles serve as a versatile tool to enforce role-based access control (RBAC) and personalize user experiences within the system. Group profiles extend this concept by centralizing authority assignments for groups of users.

Resource security is implemented through the concept of authorities, which govern the ability to access objects. The system offers finely grained authority definitions, which include subsets such as \*ALL, \*CHANGE, \*USE, and \*EXCLUDE. This mechanism applies to files, programs, libraries, and any object within the system.

Encryption is relevant for security, with IBM i enabling data encryption at the ASP and Database Column levels. However, encryption operations can be carefully managed to mitigate performance implications. Security audit journals facilitate monitoring security effectiveness to log selected security-related events for review.

In the context of *Ansible for IBM i*, the integration of security measures is accommodated through a range of purpose-built modules. These modules permit diverse requirements, which include security and compliance checks, so that administrators can configure, verify, and optimize security settings. Ansible for IBM i offers a robust ecosystem that administrators can use to help ensure security compliance by referencing the CIS IBM i Benchmark documentation and employing regularly updated security compliance playbooks. This dynamic framework contributes to the creation of secure IBM i environments that align with modern security paradigms.

## Overview of the security management use case

The content here is designed to serve the security management use case. The playbooks that are involved offer you readily available samples that can be used as-is or adapted to suit your specific requirements.

At the time of writing, the focus of these playbooks centers on security compliance checks. These playbooks are initially presented as basic examples with plans to expand their contents based on security compliance suggestions that are outlined in the CIS IBM i Benchmark documentation.

**Note:** For more information about implementing security practices on IBM i systems by using Ansible, see the provided use cases and security management resources that are available at this [GitHub repository](#). To stay up to date, regularly review this directory under the devel branch.

In this section, you see the explanation of the playbooks that are involved in this use case:

- ▶ `main.yml`: This playbook serves as an entry point that orchestrates the execution of all other playbooks that are in this directory. Running this playbook runs the entire suite.
- ▶ `manage_system_values.yml`: This playbook verifies security-related system values against recommendations from the CIS IBM i Benchmark documentation. This playbook offers two separate YAML files for checking and remediating, along with three distinct modes of operation.
  - `system_value_check.yml`: Conducts a compliance check on system values by comparing them with the expected values.
  - `system_value_remediation.yml`: Provides remediation options that are based on user input to perform remediation after a comprehensive review of the report.

- ▶ `manage_user_profiles.yml`: This playbook uses the `ibmi_user_compliance_check` module and the `ibmi_sql_query` module to assess user profile settings. It contains the following two playbooks:
  - `user_profile_check.yml`: Performs a compliance check on user profiles.
  - `user_profile_remediation.yml`: Offers suggestions for remediation and performs remediation actions that are based on user input.
- ▶ `manage_network_settings.yml`: This playbook verifies a single network attribute setting by invoking the Retrieve Network Attributes (RTVNETA) command.
- ▶ `manage_object_authorities.yml`: This playbook validates object authorities. At the time of writing, it offers a basic example that uses the `ibmi_object_authority` module.

### Additional Information

For more information about how to run a playbook that is dedicated to Secure Compliance for IBM i, see Appendix A, “Use case for security compliance on IBM Power Systems through Red Hat Ansible” in *IBM Power Systems Cloud Security Guide: Protect IT Infrastructure In All Layers*, REDP-5659. That appendix presents a detailed use case that focuses on security compliance for IBM Power servers by using Red Hat Ansible. The section “Security and Compliance with Red Hat Ansible for IBM i” shows the configuration process.

## 6.4.3 Patch management

Figure 6-8 shows an illustrative depiction of the process of IBM Patch Management that highlights its key components and how they interact.

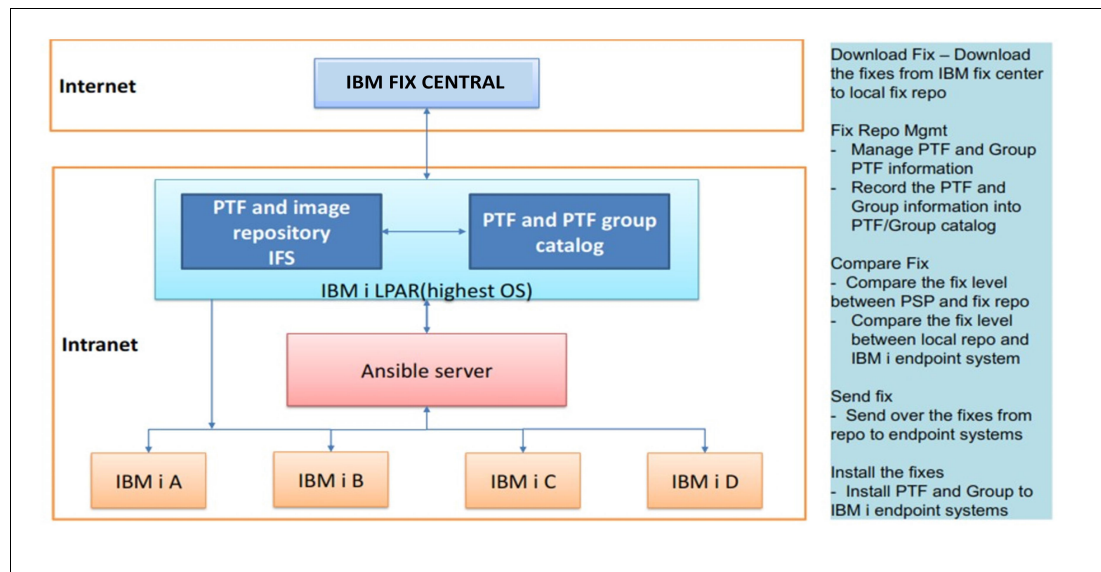


Figure 6-8 Advanced Patch Management workflow

The mechanism centers on IBM Patch Central, an online portal that establishes an internet connection with multiple IBM i instances. These instances consist of two parts:

- ▶ PTF and Image Repository IFS: Where the acquired fixes are stored and automatic detection of new Program Temporary Patch (PTF) groups is enabled.
- ▶ PTF and PTF Group Catalog: Stores information about downloaded fixes and assembles catalogs with new media.

To help ensure coherence and robustness, a meticulous comparison of PTF versions against repository content and IBM i endpoints' systems takes place. This crucial coordination is orchestrated by the Ansible Controller, which effectively connects with each component within the intranet framework. The Ansible Controller can install PTFs and thoroughly examine the status of PTFs on the endpoints.

This dynamic use case, which is equipped with comprehensive functions, is available for download and adaptation. For more information, see the GitHub repository [Patch Management](#).

## 6.4.4 Configuration tuning

In this section, you learn about a set of playbooks that revolve around IBM i services. These playbooks, which are sourced from the GitHub repository [IBM i Services](#), can optimize various aspects of an IBM i environment.

Here are the playbooks. Their purposes are explained in the following sections.

- ▶ `communications.yml`
- ▶ `message_handling.yml`
- ▶ `product_checking.yml`
- ▶ `system_health.yml`
- ▶ `work_management.yml`

### Analyzing IBM i network communications

The playbook `communications.yml` addresses the critical task of assessing and optimizing network communications within the IBM i environment. By using various tasks, administrators can use this playbook to enhance the efficiency, security, and reliability of network connections.

The playbook commences with gathering essential facts, excluding default fact collection, by using the `gather_facts: false` parameter. It uses the `ibm.power_ibmi` collection to run its tasks. The `<become_user_name>` and `<become_user_password>` variables are employed for privilege escalation.

The playbook orchestrates a series of tasks that offer in-depth insights into network communications:

1. **Review most data transfer connections:** This task employs the `ibmi_sql_query` module to retrieve connections that transfer substantial data (over 1 GB). The retrieved results are registered, which provide crucial metrics about data flow. The subsequent debug task showcases these results, which aid administrators in evaluating resource utilization and potential bottlenecks.
2. **Analyze remote IP address details for password failures:** By using SQL queries, this task identifies and counts occurrences of failed password attempts from remote IP addresses within the past 24 hours. The gathered information is registered and presented through the debug task. This analysis helps administrators detect potential security threats and unauthorized access attempts.
3. **Review TCP/IP routes:** This task uses the `ibmi_sql_query` module to pinpoint TCP/IP routes with inactive local binding interfaces. By registering and displaying the details of these routes, administrators can identify and rectify potential network configuration issues that might impact communication reliability.



## Optimizing message handling on IBM i

The playbook `message_handling.yml` can improve message handling within the IBM i environment so that administrators can proactively monitor and manage message queues and their responses. By facilitating smoother communication, the playbook helps maintain system health and performance.

With the factual data collection disabled by using `gather_facts: false`, the playbook harnesses the `ibm.power_ibmi` collection.

The playbook orchestrates a series of tasks for thorough message handling optimization:

1. Analyze next initial program load (IPL) status: This task employs the `ibmi_sql_query` module to examine history log messages since the last IPL to predict the nature of the next IPL. It assesses whether it will be normal or abnormal based on specific messages. The results are registered, and an assert task helps ensure that the next IPL is predicted to be normal, which enhances system predictability and stability.
2. Examine system operator inquiry messages with replies: This task employs SQL queries to retrieve system operator inquiry messages and their associated replies from the message queue QSYSOPR. The gathered information is registered, which offers administrators insights into system operator interactions and responses, promoting efficient communication and issue resolution.
3. Examine system operator inquiry messages without replies: By analyzing system operator inquiry messages that have not received replies, this task enhances message handling efficiency. SQL queries extract relevant data from the QSYSOPR message queue, and the results are registered and presented through the debug task.

## License and product monitoring for IBM i

The playbook `product_checking.yml` provides administrators with valuable insights into the licensing and expiration status of products within IBM i. The playbook operates under the premise that helping ensure the validity of licensed products is crucial for system health and compliance.

The playbook orchestrates two key tasks:

1. Monitoring expiring licenses: This task uses the `ibmi_sql_query` module to retrieve information about all licensed products and features that expire within the next 2 weeks. This information is crucial for proactive license management and preventing disruptions due to expired licenses. Results are registered under the `<expire_within_next_2_weeks>` variable, which provides administrators with actionable insights.
2. Monitoring expiring licenses for installed products: This task employs SQL queries to retrieve details about licensed products and features that expire within the next 2 weeks, focusing on installed products. By considering only products that are marked as `INSTALLED = YES`, administrators can prioritize active components that require license renewal. Results are registered under the `<expire_within_next_2_weeks>` variable for a comprehensive view.

## IBM i system health analysis

The playbook `system_health.yml` facilitates comprehensive system health analysis within the IBM i environment. It provides administrators with invaluable insights into system integrity and performance.

The playbook orchestrates two central tasks that are key for maintaining system health:

1. **Unofficial code inspection:** The playbook uses the `ibmi_sql_query` module to scrutinize the presence of any unofficial IBM i code within the QSYS library. By querying the `QSYS2.OBJECT_STATISTICS` view for objects that are labeled `*PGM *SRVPGM`, the playbook identifies unsigned objects in the `*SYSTEM` domain. This inspection promotes security and stability. The results are registered under the `<unofficial_code_check>` variable.
2. **Large table identification:** Another task employs SQL queries to identify tables within the `QSYS2.SYSLIMITS` view that exceed a `CURRENT_VALUE` of 10,000,000. This task highlights tables with a significant data volume, which potentially indicates performance issues. The results are captured in the `<large_table>` variable for further analysis.

## IBM i work management analysis

The playbook `work_management.yml` focuses on IBM i work management analysis to deliver enhanced insights into job queue efficiency and system performance.

This playbook uses a series of critical tasks that are geared toward efficient work management:

1. **Scheduled job evaluation:** The playbook employs the `ibmi_sql_query` module to assess job schedule entries that are no longer effective due to explicit holding or scheduling limitations. The inspection, which is centered on the `QSYS2.SCHEDULED_JOB_INFO` view, targets `HELD` and `SAVED` status entries. The results are stored under the `<job_schedule_status>` variable.
2. **Job queue and temporary storage analysis:** The playbook capitalizes on SQL queries to uncover jobs waiting to run within job queues (`QSYS2.JOB_INFO`). Furthermore, it scrutinizes the top four consumers of temporary storage based on memory pool usage, and isolates jobs with temporary storage exceeding 1 GB. These analyses contribute to optimized system resource allocation.
3. **Lock contention evaluation:** Through SQL queries, the playbook identifies jobs that encounter excessive lock contention. By querying the `QSYS2.ACTIVE_JOB_INFO` view, it isolates jobs with combined database and non-database lock waits that surpass 2000. Insights are essential for maintaining operations.
4. **QTEMP resource utilization inspection:** The playbook evaluates host server jobs that use more than 10 MB of QTEMP storage. By using `qsys2.active_job_info`, jobs meeting this criterion are identified, which enables efficient resource allocation.



## Future trends and directions

This chapter covers the future direction and plans of Ansible and IBM Power at a high level. It also covers emerging trends in Ansible automation, and how you can maintain your Ansible code with products like Visual Studio Code and IBM watsonx Code Assistant for Red Hat Ansible Lightspeed.

The following topics are described in this chapter:

- ▶ Ansible and IBM Power Roadmap
- ▶ Roadmap for Ansible automation in the Power ecosystem

## 7.1 Ansible and IBM Power Roadmap

In the past, modules that were specific to IBM Power were created by the community. However, as you can see in 1.5, “Ansible for Power” on page 30, IBM has been actively creating Ansible content for use across IBM Power servers since 2020. These collections are available on both Ansible Galaxy and Ansible Automation Hub.

IBM continues to develop new content, and improve existing content within these collections. Look at the IBM Power AIX collections at Galaxy (for example), and you can see the version release cycle that is shown in Figure 7-1.

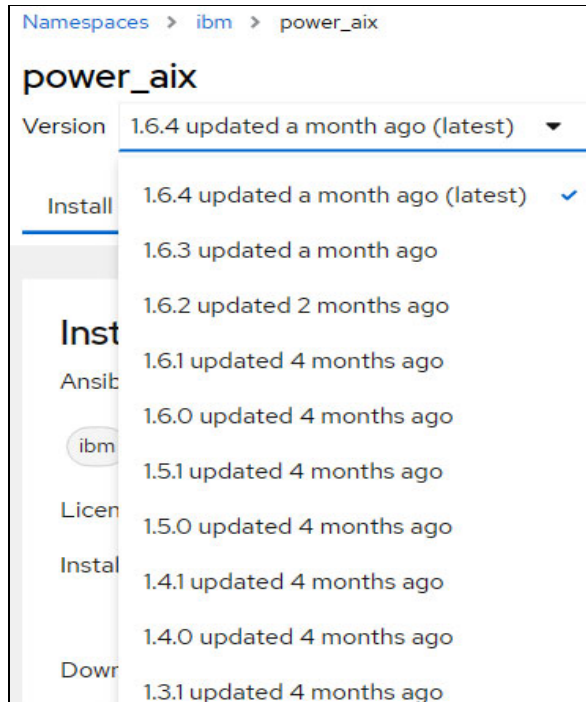


Figure 7-1 AIX collection release cycle

### 7.1.1 Working closely with the IBM Power collections and their contents

You can see what is included in the Ansible IBM Power collections and sample playbooks at the IBM GitHub pages.

#### Viewing the content of the IBM Power collections

The content within each of the IBM Power collections can be viewed on the IBM GitHub pages. The link to each specific GitHub repository can be accessed by using the Repo link within Galaxy.

Table 7-1 on page 333 provides some of those links.

Table 7-1 GitHub content for IBM Power collections

Collection	GitHub URL
ibm.power_aix	<a href="https://github.com/IBM/ansible-power-aix">https://github.com/IBM/ansible-power-aix</a>
ibm.power_ibmi	<a href="https://github.com/IBM/ansible-for-i">https://github.com/IBM/ansible-for-i</a>
ibm.power_hmc	<a href="https://github.com/IBM/ansible-power-hmc">https://github.com/IBM/ansible-power-hmc</a>
ibm.power_vios	<a href="https://github.com/IBM/ansible-power-vios">https://github.com/IBM/ansible-power-vios</a>

In the GitHub repositories, you can see the code that is used to supply the collection, including the readme file, the modules, and some sample playbooks.

## Raising an issue or suggesting enhancements to the IBM Power collection

Because the IBM Power collections are available for anyone to see and use within Ansible Galaxy, you can request new features or suggest enhancements to the existing code. You can also raise issues with the code for the development team to review. To do so, go to the Issues section of the GitHub repository for the collection. You see three options: Bug report, Custom issue template, and Feature request, as shown in Figure 7-2.

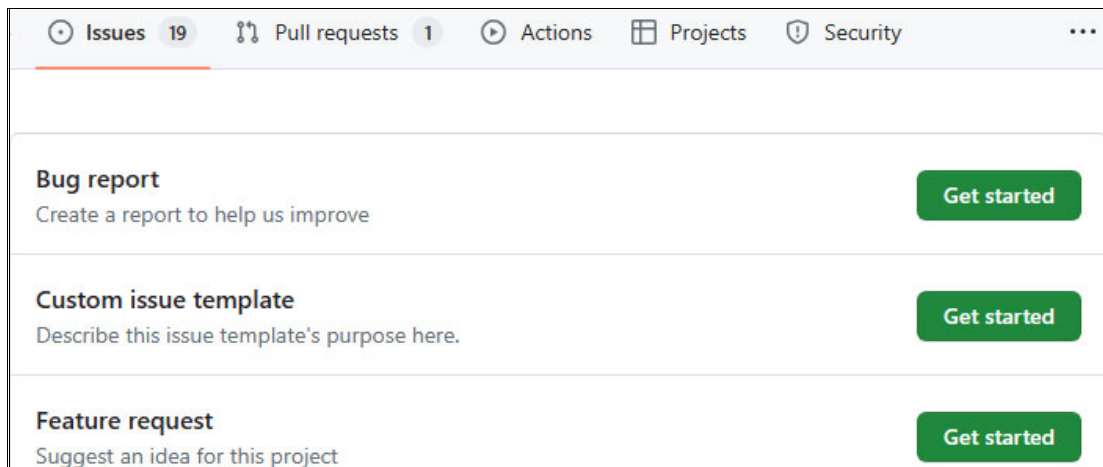


Figure 7-2 Raising a bug report or new feature request

You can also contribute to the collections by creating your own fork from the repository, making your changes to your fork, and raising a pull request. This way, the development team sees your proposed changes and either merges them into the collection or rejects them.

## 7.2 Roadmap for Ansible automation in the Power ecosystem

IBM and Red Hat continue to work on enhancements to the Ansible integrations of the IBM Power ecosystem, which includes updates to existing collections and implementations to provide more functions involving machine learning and artificial intelligence (AI).

## 7.2.1 Ansible Automation Platform on IBM Power

In June 2023, IBM and Red Hat announced that Ansible Automation Platform 2.4 was available as a technology preview on IBM Power. In December 2023, Red Hat announced the general availability of Ansible Automation Platform 2.4 on IBM Power (also announced was support for IBM Z and IBM LinuxONE). In addition to using Ansible to automate client endpoints on IBM Power (for example, AIX, IBM i, and VIOS), you can run Ansible Automation Platform and all its components on IBM Power too.

For more information about what is new in Ansible Automation Platform 2.4, including the technology preview announcement, see [What's new in Ansible Automation Platform 2.4](#), see [Red Hat Blog](#).

You can run the Ansible Controller (formerly Tower) on IBM Power and all the other components that go to make up the Ansible Automation Platform, including execution environments, event-driven Ansible, and an automation hub.

## 7.2.2 Visual Studio Code

Visual Studio Code (also known as VS Code) is an open-source code editor that is available for Windows, Mac, Linux, and a web interface. Visual Studio Code is the first GUI code editor to have an Ansible extension that is released by Red Hat.

The Ansible extension provides smart auto completion, syntax highlighting, validation, documentation reference, integration with `ansible-lint`, diagnostics, goto definition support, and command windows to run `ansible-playbook` and the `ansible-navigator` tool for both local and execution-environment setups.

Visual Studio Code can be downloaded from this [Visual Studio Code repository](#).

### Installing the Ansible extension

To install the Ansible extension, complete the following steps:

1. Open VS Code and click the **Extensions** icon in the left pane. Search for the Ansible extension that is published by Red Hat, as shown in Figure 7-3 on page 335.

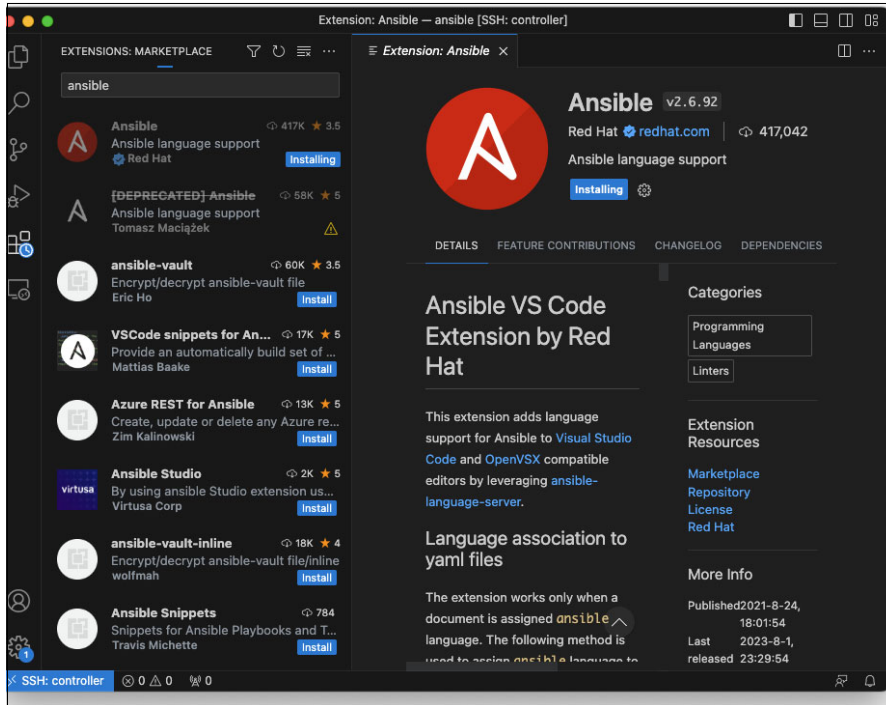


Figure 7-3 Installing the Ansible extension for Visual Studio Code

2. Open the folder that contains your Ansible files by clicking the **Explorer** icon in the upper left, as shown in Figure 7-4.

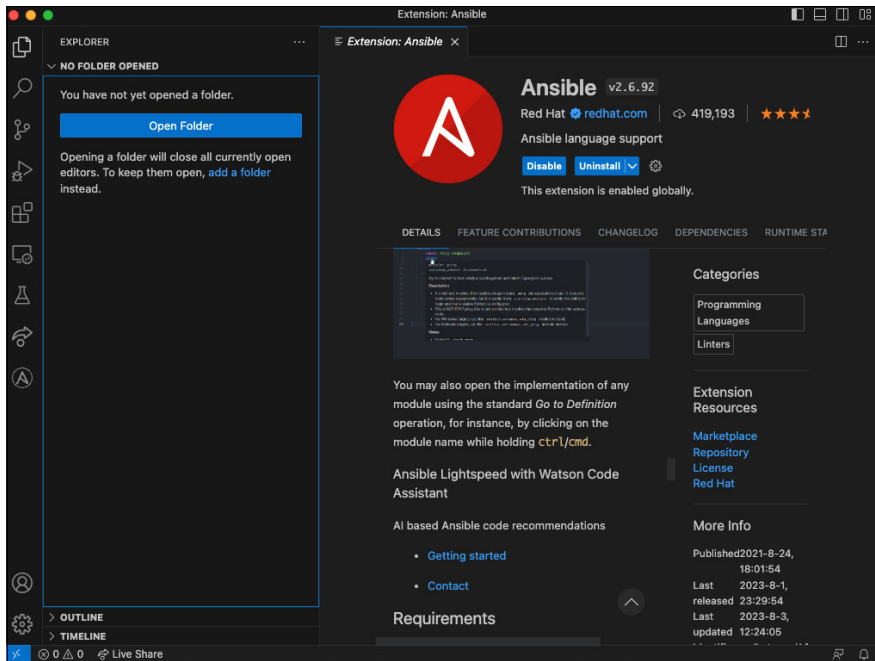


Figure 7-4 Opening the folder that contains your ansible files

3. The first time that you open an Ansible file (either `.yaml` or `.yml`), there are a couple of steps that you must do:
  - a. Define which Python environment that the Ansible extension should use by clicking the Python version indicator, which is at the right of the Status Bar, as shown in Figure 7-5.

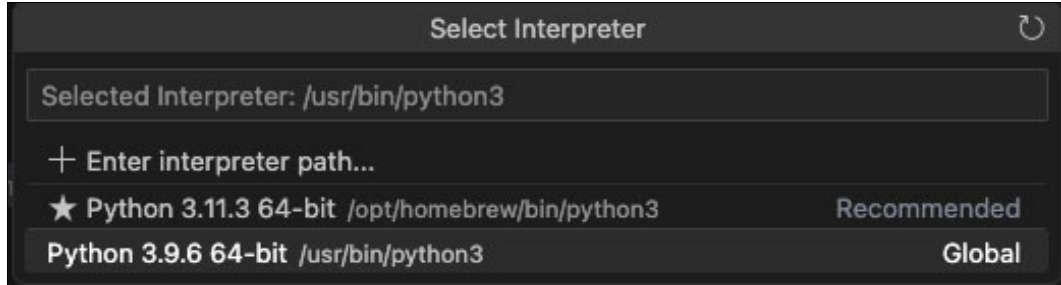


Figure 7-5 Selecting the Python environment

- b. Associate the `.yaml` or `.yml` files with the Ansible file type by clicking the language indicator, which is at the right of the Status Bar, and selecting **Ansible** from the drop-down menu, as shown in Figure 7-6. The language indicator is probably set to YAML before it is associated with the Ansible file type.

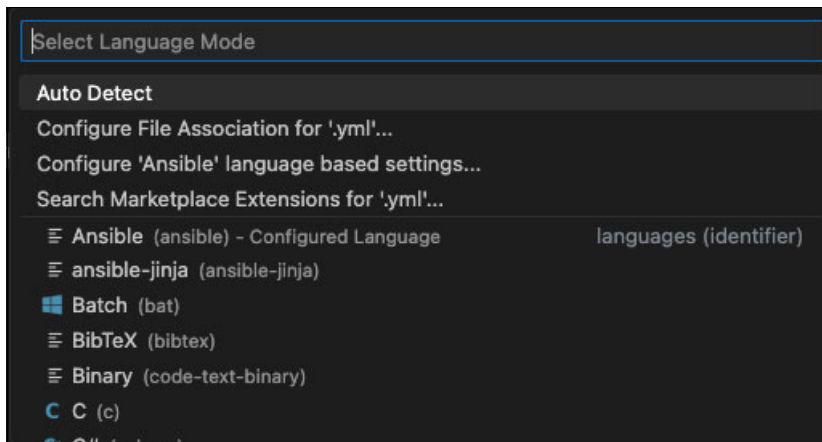


Figure 7-6 Associating YAML files with the Ansible language

The Ansible extension should now recognize YAML files as Ansible language, and offer syntax checking, documentation links, and other contextual aids to help you write Ansible code.

**Note:** If you have the `ansible-lint` package installed, it is automatically integrated for syntax checking and code validation.

If you are working in a larger environment, you probably will not write, test, and run your Ansible code on the same workstation where you installed Visual Studio Code. In this case, install the Remote Secure Shell (SSH) extension.

### Installing the Remote SSH extension

Install the Remote SSH extension by clicking the **Extensions** icon in the left navigation bar. Search for “remote” and install the “Remote - SSH” extension that is published by Microsoft, as shown in Figure 7-7 on page 337.



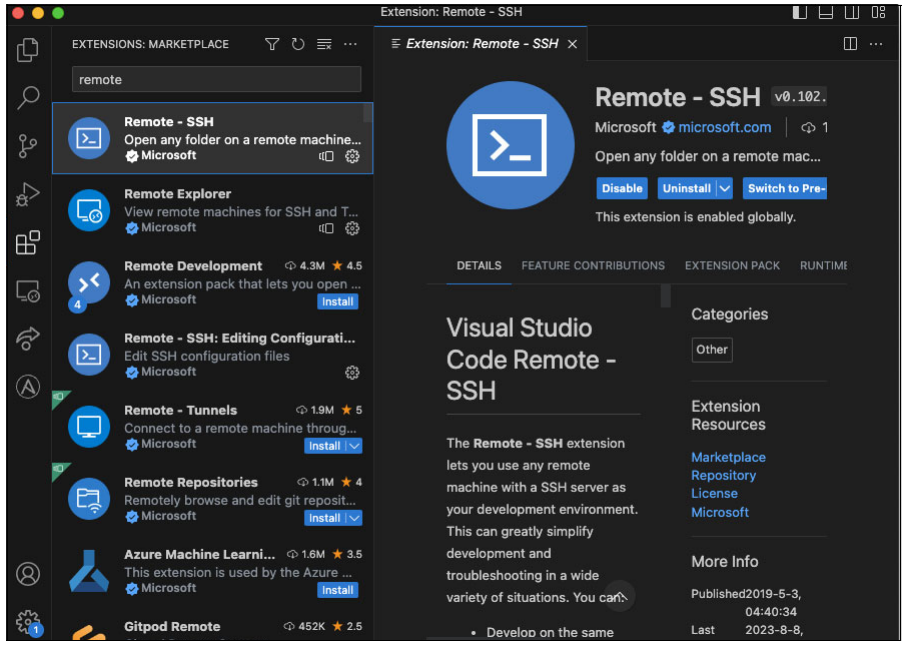


Figure 7-7 Installing the Remote-SSH extension

Once the extension is installed, open the VS Code Command Palette by pressing F1, and search for 'remote', as shown in Figure 7-8. Use either “Remote-SSH: Connect to Host...” or “Remote-SSH: Add New SSH Host” to enter the hostname and user credentials for the remote machine that you use to test and run your Ansible code.

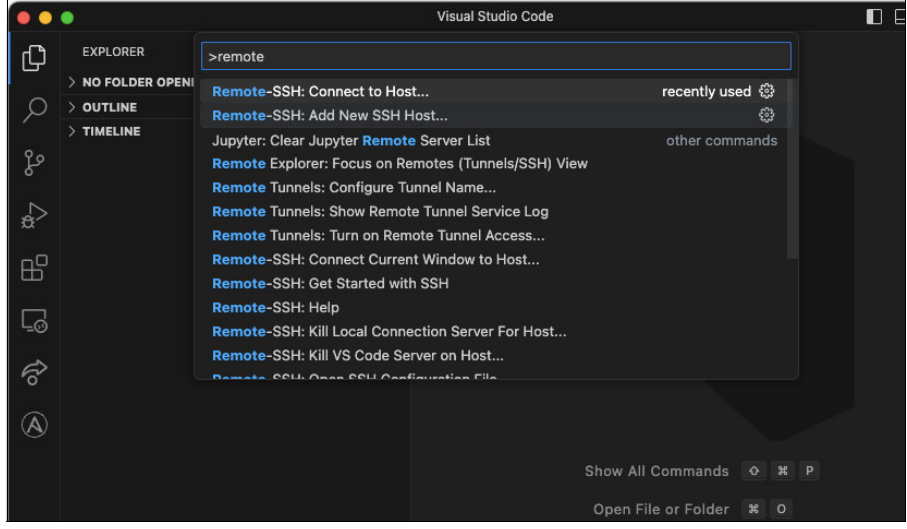


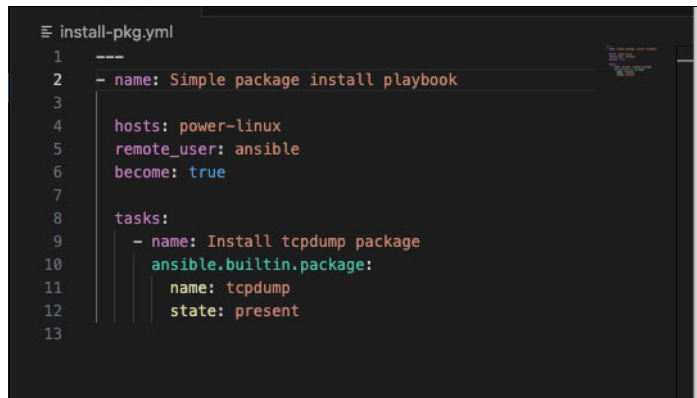
Figure 7-8 Connecting to a remote host with the Remote-SSH extension

**Contextual aids in the Ansible extension**

The Ansible extension for VS Code is designed to provide many contextual aids to writing and testing your Ansible code. This section shows some of the features of the VS Code extension.

## Syntax highlighting

Ansible module names, module options, and keywords are recognized and displayed in distinctive colors so that the developer can see whether the language syntax matches the intended purpose. Default colors change depending on the color theme that is used. An example of the Visual Studio Dark theme is shown in Figure 7-9.

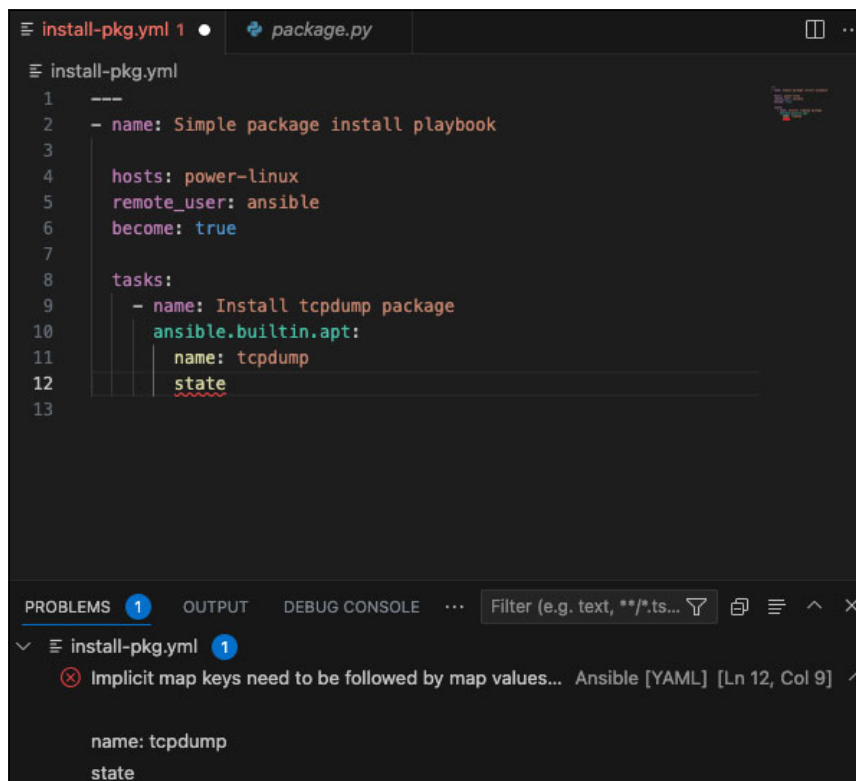


```
1 ---
2 - name: Simple package install playbook
3
4   hosts: power-linux
5   remote_user: ansible
6   become: true
7
8   tasks:
9     - name: Install tcpdump package
10       ansible.builtin.package:
11         name: tcpdump
12         state: present
13
```

Figure 7-9 Ansible code syntax highlighting

## Validation

The Ansible extension provides feedback regarding syntax as you type, and any potential problems are shown in the **Problems** tab of the integrated terminal, as shown in Figure 7-10.



```
1 ---
2 - name: Simple package install playbook
3
4   hosts: power-linux
5   remote_user: ansible
6   become: true
7
8   tasks:
9     - name: Install tcpdump package
10       ansible.builtin.apt:
11         name: tcpdump
12         state
13
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE ... Filter (e.g. text, \*\*/\*.ts...)

install-pkg.yml 1

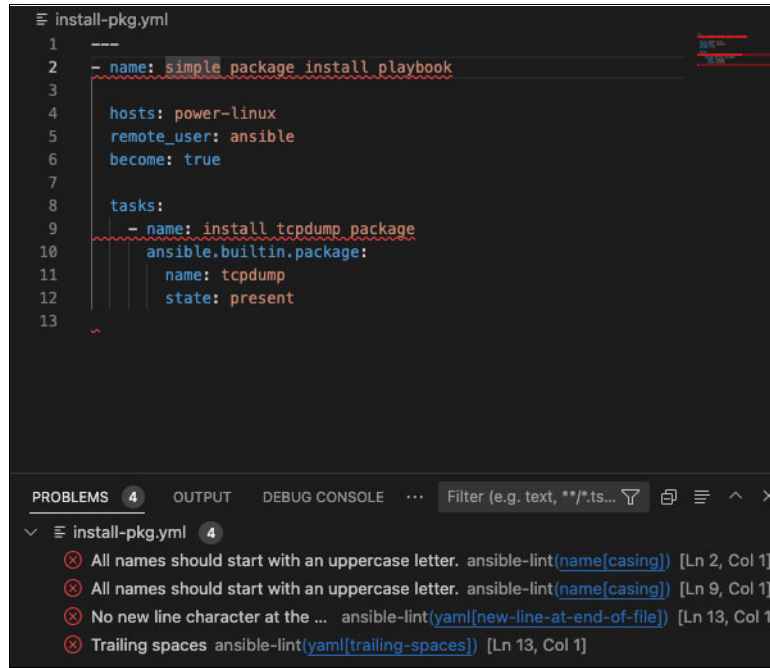
Implicit map keys need to be followed by map values... Ansible [YAML] [Ln 12, Col 9]

```
name: tcpdump
state
```

Figure 7-10 Code validation that is shown in the Problems tab

## Integration with ansible lint

When the `ansible-lint` package is installed, it is integrated into the Ansible extension. `ansible-lint` runs in the background whenever a file is opened or saved. Lines of code with errors are highlighted, and a more detailed description of the error is shown in the **Problems** tab, as shown in Figure 7-11.



```
install-pkg.yml
1 ---
2 - name: simple package install playbook
3
4   hosts: power-linux
5   remote_user: ansible
6   become: true
7
8   tasks:
9     - name: install tcpdump package
10       ansible.builtin.package:
11         name: tcpdump
12         state: present
13
```

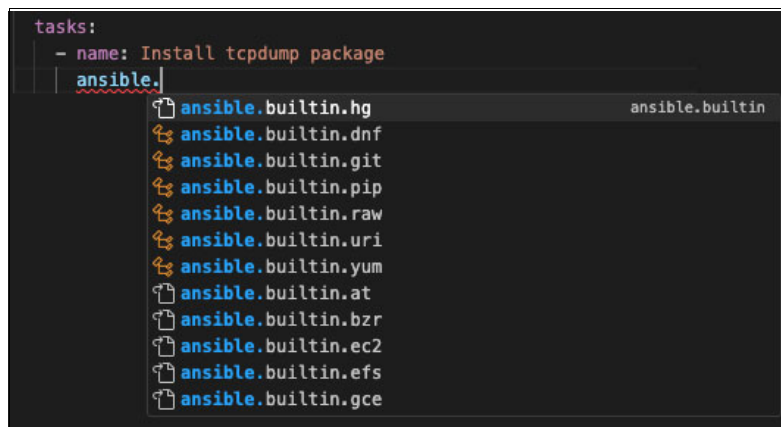
PROBLEMS 4 OUTPUT DEBUG CONSOLE ... Filter (e.g. text, \*\*/\*.ts...)

- ⊗ All names should start with an uppercase letter. `ansible-lint(name[casing])` [Ln 2, Col 1]
- ⊗ All names should start with an uppercase letter. `ansible-lint(name[casing])` [Ln 9, Col 1]
- ⊗ No new line character at the ... `ansible-lint(yaml[new-line-at-end-of-file])` [Ln 13, Col 1]
- ⊗ Trailing spaces `ansible-lint(yaml[trailing-spaces])` [Ln 13, Col 1]

Figure 7-11 Code syntax warnings that are generated by `ansible-lint`

## Smart auto-completion

As you type, the Ansible extension offers suggestions to possible options depending on the context of the code, as shown in Figure 7-12. You can select and accept a suggestion, or disregard all options.



```
tasks:
- name: Install tcpdump package
  ansible.builtin
```

- ansible.builtin.hg
- ansible.builtin.dnf
- ansible.builtin.git
- ansible.builtin.pip
- ansible.builtin.raw
- ansible.builtin.uri
- ansible.builtin.yum
- ansible.builtin.at
- ansible.builtin.bzr
- ansible.builtin.ec2
- ansible.builtin.efs
- ansible.builtin.gce

Figure 7-12 Auto-completion of a module name

### Documentation reference

Hovering your cursor over a module name, module option, or keyword shows a brief description of the item as a tool tip, as shown in Figure 7-13. You can display a full definition by right-clicking the item and selecting **Go to definition**. The full definition appears in a separate tab. Alternatively, you can select **Peek** from the menu to display the definition as a dialog box.

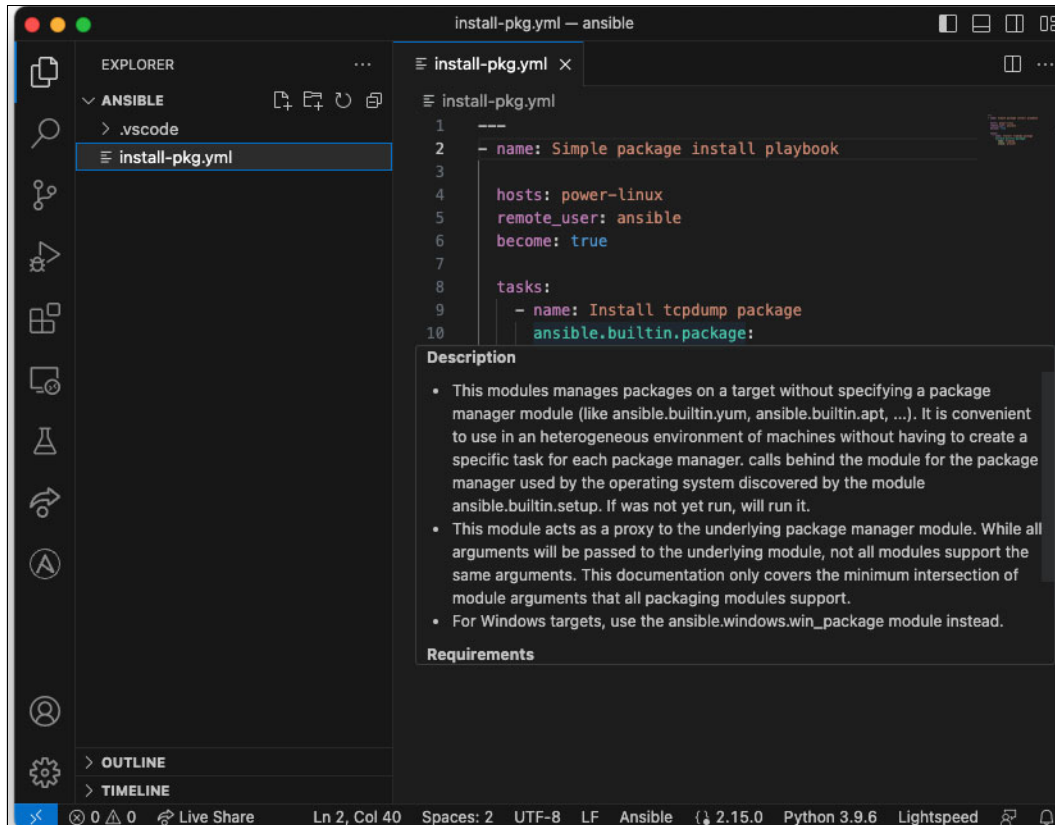


Figure 7-13 Module documentation that is shown as a tool tip

### Running a playbook in an integrated terminal

You can run a playbook from VS Code by right-clicking the playbook name in the **Explorer** tab, and then run the playbook by selecting either `ansible-navigator` or `ansible-playbook`, as shown in Figure 7-14.

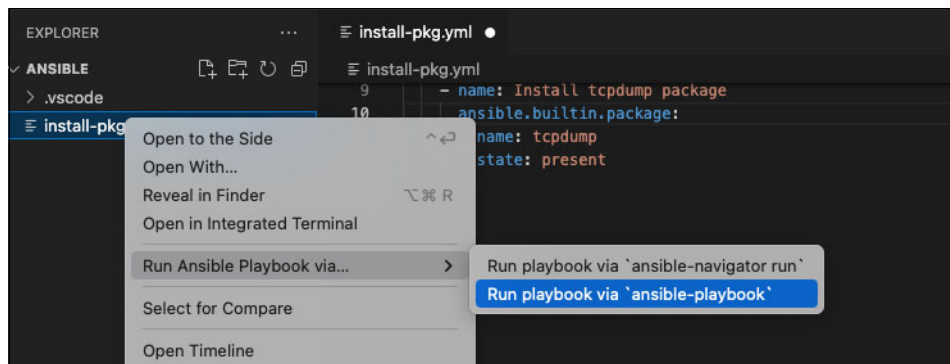


Figure 7-14 Running a playbook in an integrated terminal

## Source control with git

With `git` installed on your workstation or your Ansible Controller, you can manage your source control by using `git` from within Visual Studio Code.

Clicking the Source Control icon on the left taskbar shows an overview of changed files that might need to be updated in your GitHub repository. Clicking the icon shows the Source Control view that you can use to commit changes with a message and push to your repository, as shown in Figure 7-15.

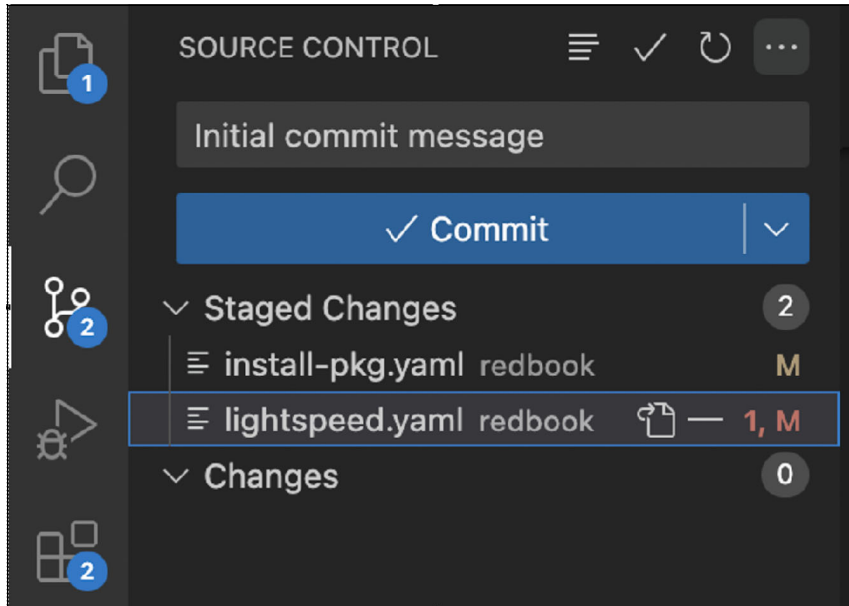


Figure 7-15 Files staged for commit to GitHub

Clicking the **Views and More Actions** menu in the upper right of the Source Control view shows more git operations, such as clone, branch, and configure remote repository, as shown in Figure 7-16.

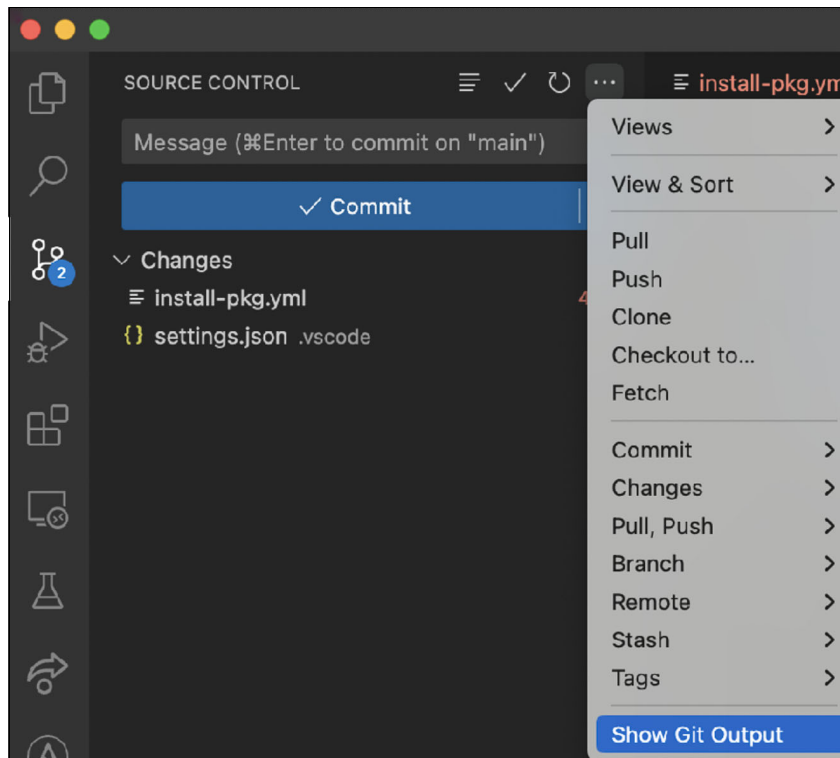


Figure 7-16 Git menu

A full description of using git in Visual Studio is beyond the scope of this book, but for more information, see the following resources:

- ▶ [Using Git source control in VS Code](#)
- ▶ [What is visual inspection?](#)

### 7.2.3 IBM watsonx Code Assistant for Red Hat Ansible Lightspeed

IBM watsonx Code Assistant for Red Hat Ansible Lightspeed is a joint project between IBM and Red Hat that offers access to Ansible content recommendations through the usage of natural language automation descriptions. This project is accessible through the integration of an IBM AI cloud service that is operated by Red Hat and the Ansible Virtual Studio Code plug-in, and is offered to the Ansible community to use without cost. This service uses, among other data, roles and collections that are available through the community website Ansible Galaxy.

IBM watsonx Code Assistant for Red Hat Ansible Lightspeed is released and available for use for Red Hat customers. At the time of writing, IBM watsonx Code Assistant for Red Hat Ansible Lightspeed does not write complete playbooks, but can generate syntactically correct and contextually relevant content by using natural language requests that written in plain English text.

## Getting started

To enable IBM watsonx Code Assistant for Red Hat Ansible Lightspeed, you need the Visual Studio Ansible Extension from Red Hat, as described in 7.2.2, “Visual Studio Code” on page 334. You also need a Red Hat login.

Once you have Visual Studio Code and the Ansible extension installed, complete the following steps:

1. Go to the Settings window for the Ansible extension, as shown in Figure 7-17.

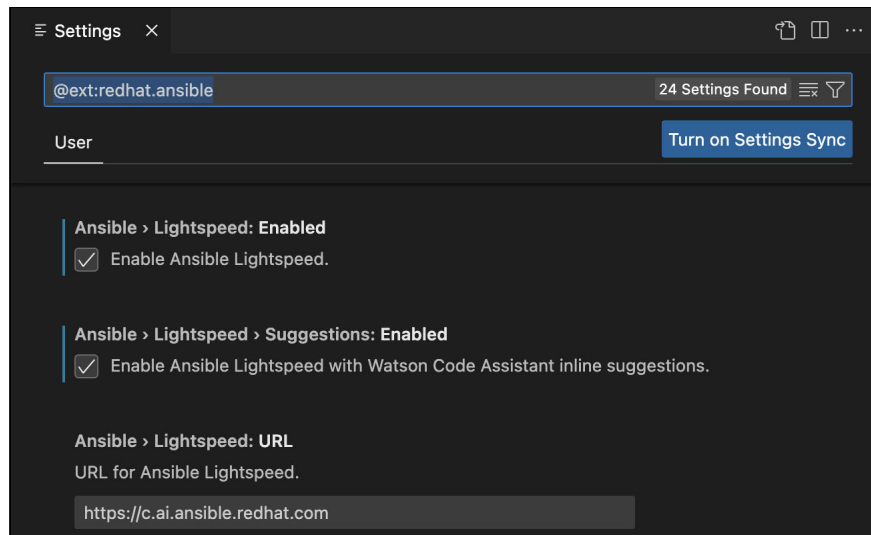


Figure 7-17 Enabling Lightspeed in Ansible extension settings

2. Enable the following settings:
  - Select **Ansible** → **Lightspeed**.
  - Select **Ansible** → **Lightspeed** → **Suggestions**.
  - Select **Ansible** → **Lightspeed**. The URL should be `https://c.ai.ansible.redhat.com`.

3. Click the **Ansible** icon (the letter A) in the left taskbar to display the Ansible Lightspeed Login window, as shown in Figure 7-18.

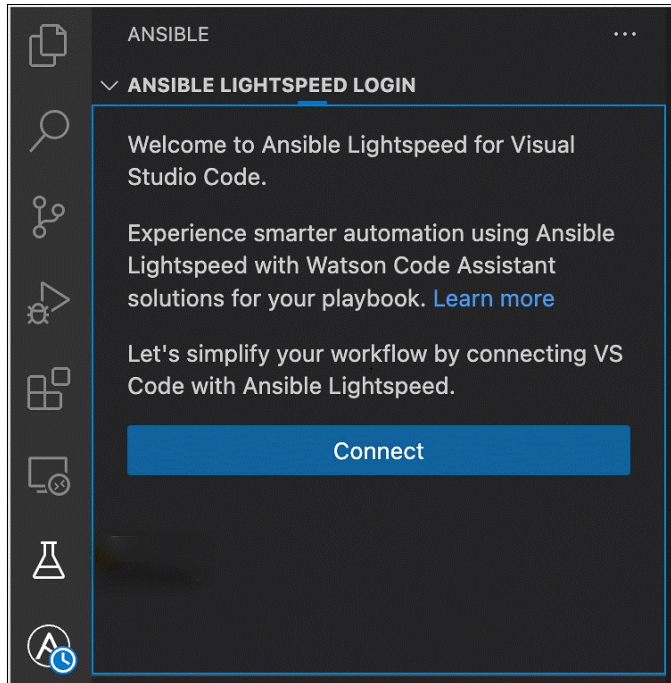


Figure 7-18 Connecting to the Ansible Lightspeed login

4. Click **Connect** in the Ansible Lightspeed Login window, and you are redirected to the IBM watsonx Code Assistant for Red Hat Ansible Lightspeed login web page. Follow the prompts to log in with your Red Hat credentials, as shown in Figure 7-19.

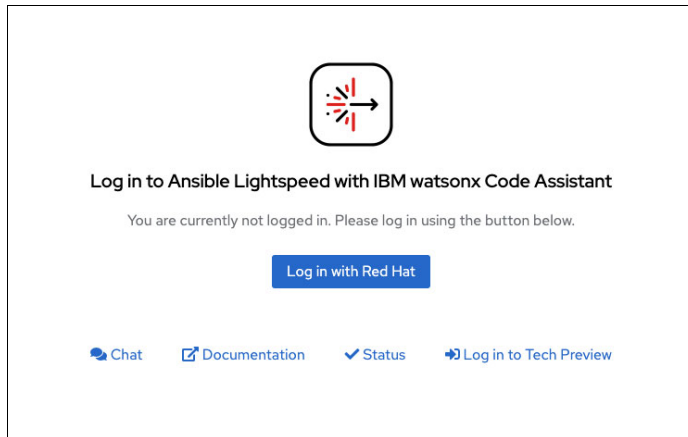


Figure 7-19 Lightspeed authentication window

5. Once authenticated, accept the Terms and Conditions to enable IBM watsonx Code Assistant for Red Hat Ansible Lightspeed.
6. Authorize VS Code to interact with IBM watsonx Code Assistant for Red Hat Ansible Lightspeed extension by sending prompts and receiving code suggestions, as shown in Figure 7-20 on page 345. You should see in the left taskbar that you now are logged in to IBM watsonx Code Assistant for Red Hat Ansible Lightspeed.



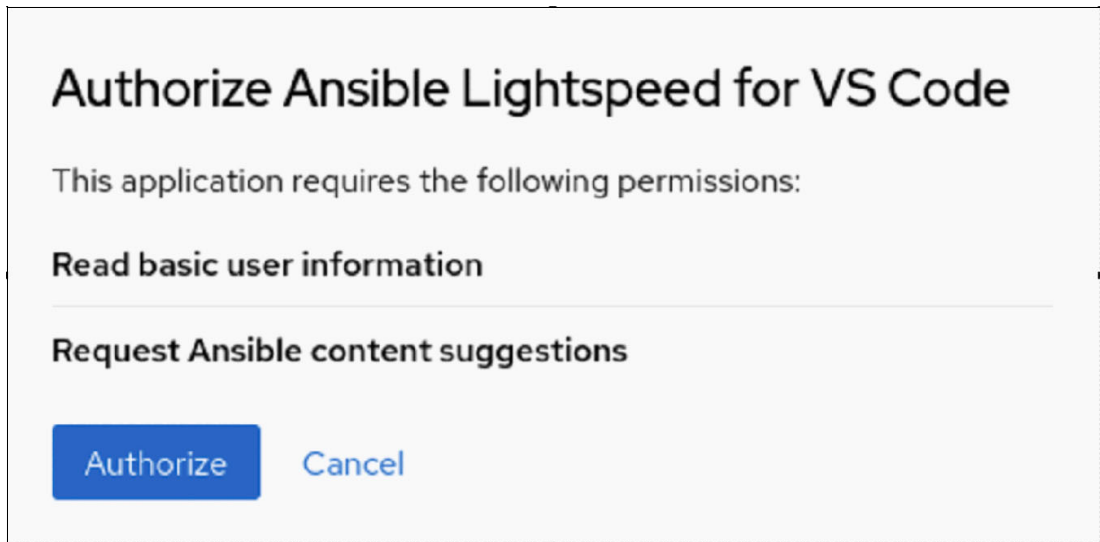


Figure 7-20 Authorizing Ansible to interact with Ansible Lightspeed with Watson Code Assistant

### Using IBM watsonx Code Assistant for Red Hat Ansible Lightspeed

To use IBM watsonx Code Assistant for Red Hat Ansible Lightspeed to get code recommendations for Ansible tasks, open a valid Ansible YAML file in the code editor. Check the bottom status bar of VS Code to help ensure that the YAML file is recognized as the Ansible language and that Lightspeed is enabled.

Enter a task name and a description of what you want the task to do. Press Enter at the end of the line, and you should receive a code suggestion, as shown in Figure 7-21.

```
55 | - name: Copy logrotate.conf template to /etc/logrotate.conf
56 |   ansible.builtin.template:
      src: logrotate.conf.j2
      dest: /etc/logrotate.conf
      owner: root
      group: root
      mode: 420
```

Figure 7-21 Code suggestion from Lightspeed

The code suggestion is shown in a gray font. Review the suggested code and either press Tab to accept the recommendation, or press Esc or Enter to close it.

The source of the code suggestion is shown in the **Ansible: Lightspeed Training Matches** tab of the pane below the code editor window, as shown in Figure 7-22.

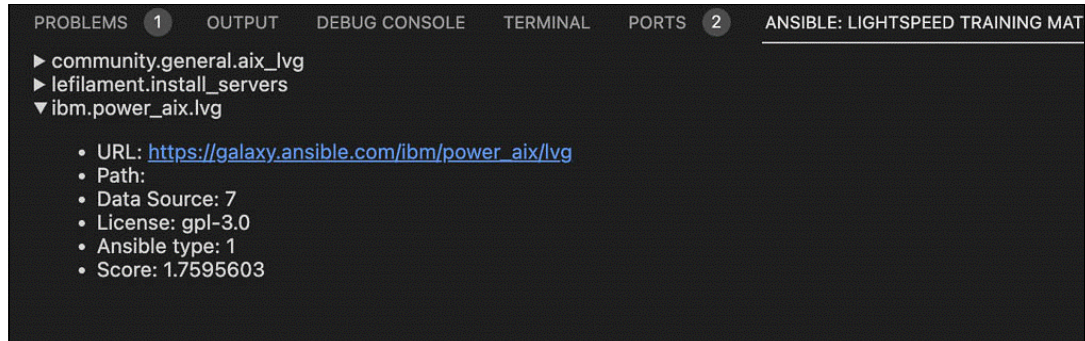


Figure 7-22 Source of code suggestion from Lightspeed

**Note:** The pane below the code editor shows various tabs: **Problems**, **Output**, **Debug Console**, and **Integrated Terminal**. The pane can be toggled by pressing Command-J (Mac) or Control-J (Windows/Linux).

### ***What happens when you reject or accept a suggestion***

The actions that you take when a recommendation is provided impact the training process of the model. If you hit the Esc key to reject a recommendation, then the telemetry process considers that a rejection of the recommendation. Red Hat and IBM view that action as the user determining that the recommendation was not suitable for their task intent. If you hit Enter and accept the recommendation, then the telemetry process considers that action an acceptance of the recommendation. Red Hat and IBM view that action as the user determining that the recommendation was appropriate to use instead of typing an alternative directly.

If a recommendation is accepted and then further edits are performed, then the act of changing the recommendation to something else is considered a modification of the recommendation. This action tells Red Hat and IBM that the recommendation required extra action to meet the intended use. This information is used for context in training the model for similar prompts in the future.

The telemetry data is first anonymized and then sent whenever you switch to a different file in Visual Studio Code or create an Ansible task in the same Ansible Playbook. For more information, see [Getting a Recommendation](#).

### ***Improving the recommended guidance***

To improve the likelihood of a quality recommendation, follow these guidelines:

- ▶ Help ensure that your YAML is properly formatted.
- ▶ Avoid context switching within a single playbook file. Lightspeed attempts to correlate earlier tasks to the active recommendation, and context switching might lead to incorrect recommendations.
- ▶ If you do not get a recommendation that aligns with the intent of your task name, then rewording your statement to provide more information about what you want might lead to different results. Try adding or removing context to see whether you get a better response.

Some example tasks to try out in IBM watsonx Code Assistant for Red Hat Ansible Lightspeed are shown in Example 7-1 on page 347.

### *Example 7-1 Example tasks for Ansible Lightspeed*

---

- name: Update all packages
  - name: Create a user named 'oracle'
  - name: Create a 40 Gb Logical Volume
  - name: Run 'uptime' command on remote servers
  - name: Create AIX volume group named datavg
  - name: Restart chronyd daemon
  - name: Add host entry to the /etc/hosts file
  - name: Add the line "Defaults logfile=/var/log/sudo.log" to /etc/sudoers
- 

For more information, see [Improving the Recommended Guidance](#).

### ***Matching recommendations to training data***

The IBM watsonx Code Assistant for Red Hat Ansible Lightspeed machine learning model is trained on content from Ansible Galaxy and other sources.

Because of the nature of deep learning technology and the kinds of content that is used to train Lightspeed, it is not possible to identify specific training data inputs that contributed to particular Lightspeed output recommendations. Nevertheless, Lightspeed includes a feature to help users that are interested in understanding the possible origins of generated content recommendations. When Lightspeed generates a recommendation, it attempts to find items in the training data set that closely resemble the recommendation. In such cases, Lightspeed displays licensing information and a source repository link for the training data matches in a window interface in the VS Code extension.

This feature might enable users to discover open-source license terms that are associated with related training data. This feature is implemented even though it is unlikely that either the training data that is used in fine-tuning or the output recommendations themselves are generally protected by copyright, or that output reproduces training data content that is controlled by copyright licensing terms.

Red Hat does not claim any copyright or other intellectual property rights in the suggestions that are generated by the IBM watsonx Code Assistant for Red Hat Ansible Lightspeed service. For more information, see [Matching Recommendation to Training Data](#).





# Unveiling IBM i Modernization Engine for Lifecycle Integration

In this appendix, you learn about the IBM i Modernization Engine for Lifecycle Integration (IBM i Merlin). This appendix provides insightful descriptions and in-depth descriptions of IBM i Merlin. The sections offer an introduction and comprehensive overview of IBM i Merlin key aspects, benefits, collaboration between IBM and ARCAD, components, content, and its significance in meeting the demands of DevOps on IBM i. This appendix aims to provide a clear understanding of IBM i Merlin and its role in modernizing the IBM i ecosystem.

The following topics are described in this appendix:

- ▶ Introduction
- ▶ What is IBM i Merlin
- ▶ IBM i Merlin: Problem-solving capabilities
- ▶ Benefits of IBM i Merlin for IBM i modernization
- ▶ Decades of collaboration: IBM and ARCAD
- ▶ Components of IBM i Merlin
- ▶ Comprehensive overview of IBM i Merlin content
- ▶ Ansible integration for IBM i lifecycle management through IBM i Merlin
- ▶ The business demands for DevOps on IBM i
- ▶ IBM i Merlin for IBM i developers
- ▶ IBM i Merlin requirements

# Introduction

In today's rapidly evolving business landscape, IBM i customers face the pressing need to modernize their applications and stay ahead of the game. As you plan your IT environment, we understand the top concerns that you grapple with, from “cobbling together” various tools to “force-fitting” IBM i native file systems. Many are still reliant on outdated technologies that lack automated change control and project builds, and grapple with monolithic, non-modular designs and ancient source editors.

IBM i Next Gen Apps are applications that can quickly respond to business needs through DevOps, CI/CD, and Agile methodologies. With a focus on encapsulating processes and data, you create assets for the business by blending technology to achieve the best fit for purpose. Moreover, you can easily incorporate new technologies, even if they are not currently “in-house.”

To get to IBM i Next Gen, you must address various challenges:

- ▶ Converting fixed-format Report Program Generator (RPG) language to free format
- ▶ Understanding and managing high volumes of code
- ▶ Refactoring mega-programs into modules
- ▶ Helping ensure intelligent builds among spaghetti code

Exposing embedded logic as services and adopting a “service consumption” mind set and tools are crucial. Using modern tools such as Git for common source code management can be transformative.

IBM i offers a range of modernization technologies to bridge the gap. Modern RPG and its integration with contemporary development tools helps address the talent gap. Connectivity with cloud-based and containerized applications through Rest APIs facilitates communication between systems. You can use independent software vendor (ISV) and open-source tools to modernize source code and adopt DevOps practices.

One such tool is IBM i Modernization Engine for Lifecycle Integration (IBM i Merlin), which is an innovative set of Red Hat OpenShift based tools to guide software developers in modernizing IBM i applications and development processes. Running in Red Hat OpenShift containers, IBM i Merlin provides a multi-platform DevOps implementation. The framework simplifies the adoption of DevOps and CI/CD practices while using technologies that promote services-based software through RESTful interface connections and enterprise message technologies.

The IBM i Merlin platform includes IBM i virtual machine (VM) management, which provisions, manages, and deletes IBM i VMs through PowerVC or Power Virtual Server in IBM Cloud. One of the actions that you can run on the IBM i server is Enable Ansible environment, which helps initiate the yum, Python, and Ansible packages.

## What is IBM i Merlin

IBM i Merlin enables a fluid generational transition that preserves customer investment while propelling the platform's evolution. The framework guides and simplifies the usage of the tools that help implement DevOps and continuous integration (CI) and continuous delivery (CD) (CI/CD). IBM i Merlin embraces standardization, making it accessible to the younger generation that is familiar with these tools. The RPG converter of IBM i Merlin is a pivotal tool that enables the modernization of core RPG code. IBM i Merlin also supports cloud infrastructure migration for agile dev and test environments.

Figure A-1 shows the IBM i Modernization Engine for Lifecycle Integration GUI. This interface works well with hybrid cloud work tools. These tools facilitate modern development and deployment of IBM i native applications by using standardized cloud methods. The GUI provides accessible and efficient solutions for application modernization and integration in the dynamic IT landscape.

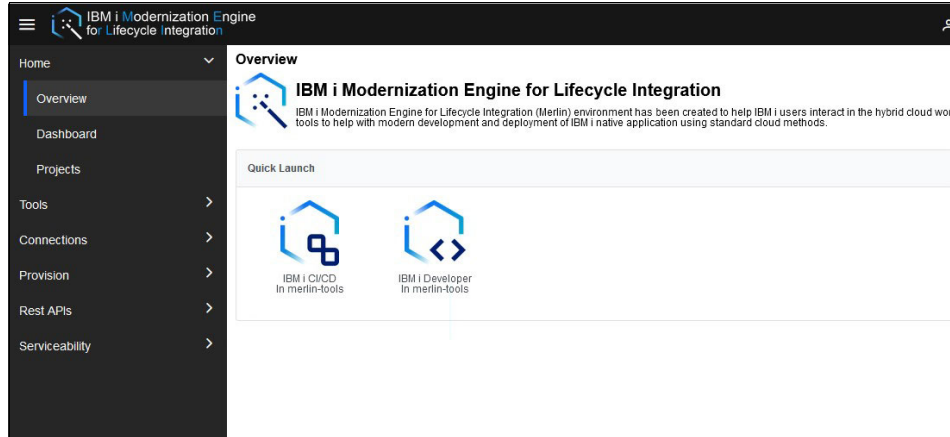


Figure A-1 IBM i Merlin GUI overview

## The role of IBM i Merlin in the IBM i market

IBM i Merlin provides a focused modernization approach. This innovative solution guides clients through the process of migrating to modern versions of existing technology.

IBM i Merlin strategically emphasizes adopting modern tools and processes, such as DevOps, cloud services, and hybrid cloud solutions to migrate the IBM i ecosystem into a new era of efficiency and adaptability. Through the integration of container-based tools, clients can keep pace with the ever-evolving demands of the market.

IBM i Merlin takes center stage in the Red Hat OpenShift conversation, positioning IBM i directly at the forefront of modernization discussions. Its containerized architecture opens doors for clients to use the potential of a hybrid cloud and multi-platform DevOps implementation to their advantage.

A crucial factor to the development of IBM i Merlin was the active involvement of IBM i customer advisory councils to help ensure that the solution aligns with the specific needs and aspirations of clients. Expert minds in the IBM i modernization domain stand ready to help clients in adopting IBM i Merlin to help ensure a successful migration.

## IBM i Merlin: Problem-solving capabilities

This section describes the problem-solving capabilities of IBM i Merlin. Table A-1 shows the impact that IBM i Merlin has across four areas.

*Table A-1 Key aspects of IBM i Merlin: Problem-solving impact*

<b>Modern / Centralized source control and branching</b>	<b>Modern RPG – modular and free format</b>	<b>Browser-centric integrated development environment based on VS Code</b>	<b>Application blueprint</b>
Use GitHub, GitLab, Bitbucket, or Gitbucket to enhance source control efficiency and facilitate efficient branching processes for improved development workflows.	Transformation of RPG code from patched to free format for enhanced modularity and readability through refactoring.	Explore the IBM i Merlin browser-centric integrated development environment (IDE) with features such as outline view, tokenization, content assist, code formatting, and language understanding.	Use IBM i Merlin capabilities for impact analysis, program understanding, data usage analysis, and program flow visualization, which helps ensure informed decisions and application integrity.

## Benefits of IBM i Merlin for IBM i modernization

Table A-2 explores the multitude of benefits that are facilitated by IBM i Merlin.

*Table A-2 Benefits of IBM i Merlin for IBM i modernization*

<b>Benefit</b>	<b>Description</b>
Faster provisioning	Experience the rapid provisioning and deprovisioning of IBM i development environments to help ensure nimbleness in your development cadence.
Modernized IBM i applications	Automate the conversion of patched-format RPG code into the more supple free form RPG to modernize your applications for heightened legibility and manageability.
Reduced time to market	Expedite the creation of IBM i business applications and realize swifter deployment to promptly address market dynamics and out pace competitors.
Single DevOps pipeline	Simplify your application development process through a unified DevOps pipeline that efficiently guides your code from testing to production realms.



Benefit	Description
Accelerated developer onboarding	Minimize the learning curve for new developers by harnessing modern tools such as Git and Jenkins to help ensure swifter adaptation and heightened productivity.
Cloud-enabled	Adopt a hybrid cloud approach and enable your IBM i applications to use the potential of a multi-platform CI/CD implementation to lead to enhanced scalability and innovation.

## Decades of collaboration: IBM and ARCAD

In 2003, ARCAD integrated with IBM WebSphere Development Studio Client (the predecessor of IBM Rational® Developer for i) to lay the groundwork for future endeavors. In 2007, ARCAD introduced the RPG Free Form converter, which revolutionized RPG development on IBM i.

In 2013, ARCAD licenses became available in IBM Passport Advantage®, which completed IBM Engineering Workflow Management integration. In 2016, ARCAD integrated with Urbancode to enhance their DevOps capabilities for IBM i. In 2017, ARCAD Observer and RPG converter were integrated into the e-config Channel. ARCAD is integrated with industry-leading DevOps flagship products such as Git (GitHub, Bitbucket, and Gitlab), IBM Engineering Workflow Management, Jenkins, and Jira.

## Components of IBM i Merlin

In the domain of IBM i modernization, IBM i Merlin acts as a potent catalyst. It introduces a collection of robust components that reshape the realm of application development. These components are purposefully crafted to facilitate smoother workflows, foster better collaboration, and embrace contemporary software practices. In this section, you learn about the pivotal components of IBM i Merlin and their significance in propelling IBM i into a new era of innovation.

The following individual components empower IBM i Merlin to excel in both problem-solving and modernization:

- ▶ Eclipse Theia

At the heart of the IBM i Merlin development environment is Eclipse Theia, which is an open-source iteration of Microsoft Visual Studio Code (VS Code). This dynamic platform offers a versatile IDE for crafting and refining IBM i applications. For more information about IDE, see “Summary of the integrated development environment” on page 354.

- ▶ Eclipse Che

A pivotal element in the IBM i Merlin infrastructure is Eclipse Che, which provides the workplace server that is responsible for crafting, managing, and orchestrating the IDE within a Kubernetes environment. This integration helps ensure a fluid and efficient development process that is further enhanced by Kubernetes orchestration.

- ▶ **ARCAD Transformer**  
Integral to the IBM i Merlin ecosystem, Transformer (formerly known as ARCAD Converter) exemplifies the powerful software that facilitates the conversion and transformation of existing code into more contemporary and efficient forms. This essential tool simplifies the modernization process and contributes to the advancement of IBM i applications.
- ▶ **ARCAD Builder**  
A cornerstone of IBM i Merlin capabilities, the ARCAD build management software (Builder) empowers developers with advanced tools for efficiently assembling and managing application components. This software promotes consistency, reliability, and efficient deployment practices throughout the development lifecycle.
- ▶ **ARCAD Observer**  
Within the IBM i Merlin toolkit, the Observer part of ARCAD emerges as a noteworthy software component. This tool provides comprehensive insights into the application development and deployment processes, which offers valuable visibility and control over critical aspects of the development lifecycle.
- ▶ **Integration with Git**  
IBM i Merlin integrates with Git, a widely adopted version control system (VCS), which enhances collaboration and code management. This integration streamlines code repository operations and aligns IBM i Merlin capabilities with modern development practices.
- ▶ **Jenkins**  
A crucial component, Jenkins facilitates CI/CD pipelines, which are a central focus of the IBM i Merlin development approach. IBM incorporated Jenkins as part of IBM i Merlin to enhance the platform's ability to automate and optimize the application delivery pipeline.
- ▶ **Red Hat OpenShift**  
IBM i Merlin finds its home and operational foundation within Red Hat OpenShift, which is a robust container platform. Red Hat OpenShift helps install, manage, and run IBM i Merlin efficiently.
- ▶ **IBM Cloud Pak® foundational services**  
IBM Cloud Pak foundational services constitute the cornerstone of the Cloud Pak ecosystem by encompassing critical tools such as Certificate Manager for efficient certificate administration to enable secure connections, and License Manager for centralized software entitlement tracking to enhance licensing efficiency. These services underscore a commitment to a robust and secure cloud environment.

**Note:** Customers do not need to pay extra to acquire the ARCAD functions. These functions are integrated into the IBM i Merlin product. As a result, developers have access to “Patched to Free” format conversion, an integrated impact analysis tool, and the capability to use intelligent build support directly within the IDE. These valuable features that are powered by ARCAD are fully integrated and included as essential components of the IBM i Merlin solution.

### Summary of the integrated development environment

In the context of IBM i modernization, an integral aspect lays in the robust IDE that effectively combines offerings from both IBM and ARCAD. This cohesive environment enhances the development process and fosters efficient modernization of applications.

Table A-3 presents a comprehensive overview of the IDE that highlights its diverse features and capabilities that help enable developers on the journey toward innovation.

Table A-3 Comprehensive overview of the integrated development environment

IBM i integration	ARCAD integration
Integrated without disruption with Code Ready Work Spaces.	Efficient Git repository setup to enable code migration from a previous library to Git.
VS Code plug-ins offer features such as tokenization, color coding, outline view, content assist, refactoring, intelligent formatting, and more.	Intelligent build metadata population.
Robust project explorer facilitates efficient IBM i environment and source management.	Conversion of code to fully free form to use deep expertise in migrating source control from an existing library to a Git-centric one.
Intelligent build with integrated compile feedback, defined metadata, and a comprehensive job log explorer.	ARCAD dependency-based build to address the complexities of IBM i applications through automated tools and processes.
Git integration for using Git-based tools, which encompass actions such as pull, push, and merge.	ARCAD impact analysis, which offers valuable insights into application linkages, data usage, and code flow visualization.

## Comprehensive overview of IBM i Merlin content

In Figure A-2, the IBM i Merlin enhanced interface is highlighted in the left pane. The components of the modernization engine are within the yellow box. This interface features the IDE and CI/CD components in a cohesive environment. Recognizing the learning curve for IBM i customers new to Linux tools and CI/CD, the platform was designed with simplicity and integration. It aims to preserve the IBM i approach by enabling a smooth migration to Linux and CI/CD for IBM i users. The framework works wherever Linux does, such as on-premises, in IBM Power servers, or other configurations. Regardless, the browser interface helps ensure a unified experience. This interface required creating a customized GUI and an engine that facilitates effective communication between code components.

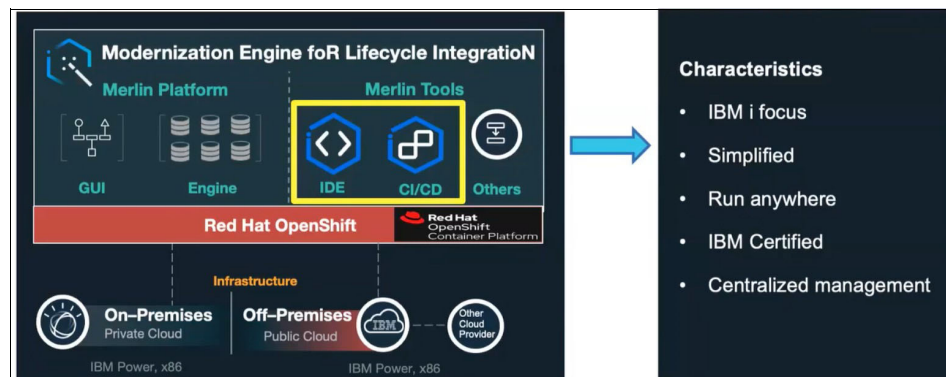


Figure A-2 Graphical view of IBM i Merlin

Key attributes:

- ▶ A modernization platform guiding IBM i applications toward hybrid DevOps.
- ▶ Exposing IBM i native functions through RESTful interfaces and centralizing IBM i connections management.
- ▶ Facilitating the use of tools for DevOps and services-based software implementation.

## Deploying the IBM i Merlin Platform through the Red Hat OpenShift Web Console

In terms of visibility, this process is mostly transparent to users. However, the underlying mechanism involves fitting IBM i Merlin into the deployment and acquisition strategies and aligning with the procurement and deployment of container-based applications from IBM. The OperatorHub serves as the platform where you can access and download IBM i Merlin while also facilitating the necessary purchases.

Figure A-3 shows this process and the OperatorHub under Red Hat OpenShift for IBM i Merlin.

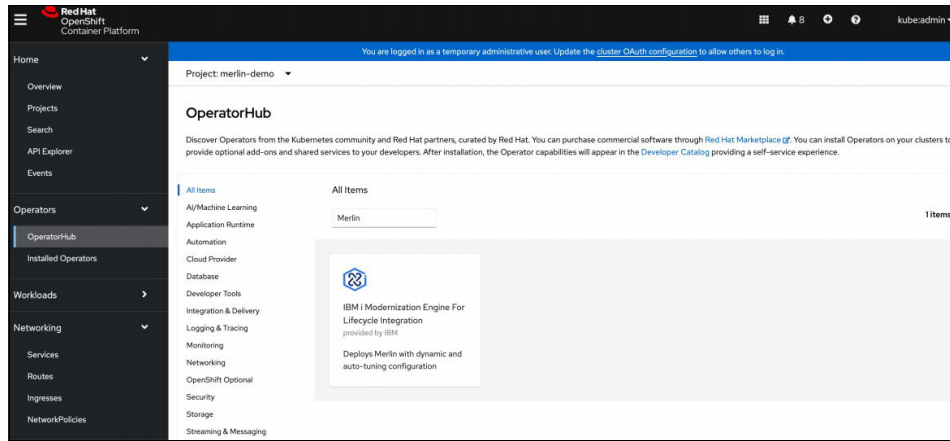


Figure A-3 OperatorHub integration with Red Hat OpenShift for IBM i Merlin

## Overview of the IBM i platform

Once deployed, a comprehensive suite of components coalesce to form a cohesive project and product. This suite includes user management, activity monitoring, and authorization enforcement to deliver the expected integrated experience of an IBM i product. What sets it apart is that it is orchestrated by a set of containers that orchestrate the IDE, CI/CD, and other elements.

Figure A-4 on page 357 shows the extensive capabilities of the IBM i Merlin platform, which include the IBM i Merlin tool lifecycle, authentication, certification management, user management, monitoring, inventory management, credential management, IBM i VM management, and IBM i software installer.

Also, the IBM i Merlin layer showcases the IBM i Merlin platform's composition, which includes both the GUI and engine. The IBM i Merlin tools are further depicted, incorporating the IDE, CI/CD, and other essential elements.

Also, the infrastructure is depicted, demonstrating where Red Hat OpenShift operates, whether on-premises or in the cloud, in ppc64 or x86 environments.

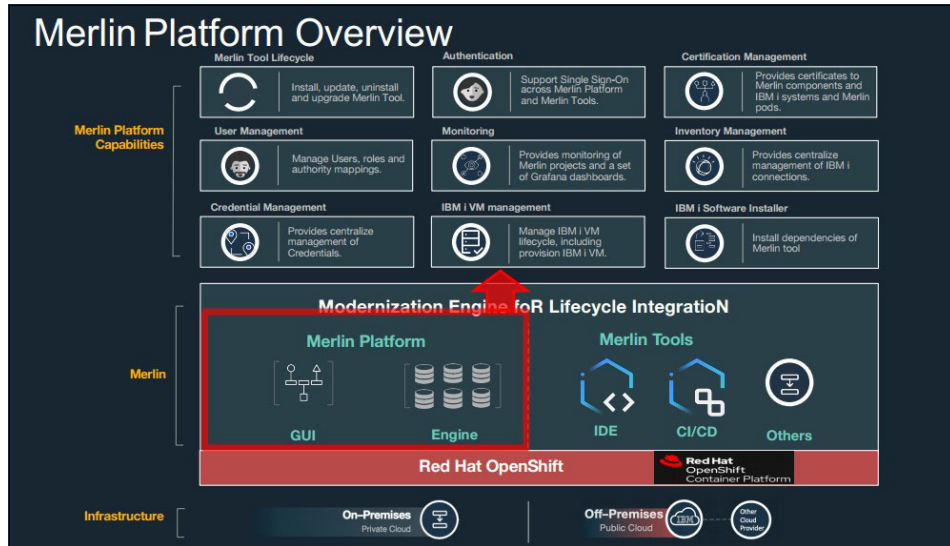


Figure A-4 IBM i Merlin platform overview

## IBM i Merlin platform: Creating RESTful services for IBM i systems

IBM i Merlin platform offers a phased approach to generate RESTful services from your existing assets so that you can target specific components for transformation. Once these components are prepared, they can be converted into services. IBM i Merlin understands the intricacies of modernization, aiming to simplify your journey.

Key points:

- ▶ A GUI for REST services: The IBM i Merlin platform provides an intuitive interface for starting RESTful service creation.
- ▶ Develop RESTful services: Create RESTful services for IBM i programs and data on IBM i systems in the initial release.
- ▶ Native IBM i execution: RESTful services continue to run natively on IBM i.
- ▶ Wide language support: Support for RPG, COBOL, and program/service program (PGM/SRVPGM).
- ▶ Data integration: Integrate data that is stored in Db2 for IBM i into your RESTful services.

Figure A-5 shows the creation of a Web Services server that is based on IBM i objects, including RPG and COBOL programs, alongside SQL statements.

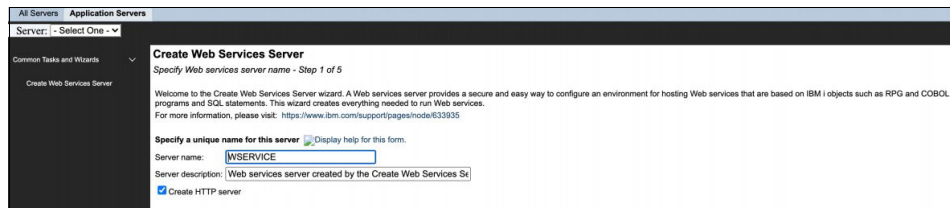


Figure A-5 Web Services server creation on IBM i objects

# Ansible integration for IBM i lifecycle management through IBM i Merlin

The lifecycle management of the product itself is adeptly handled, extending to deployment options. For example, deployment can occur in IBM Power Systems Virtual Server, which operates on IBM Cloud. Alternatively, you can deploy within your own infrastructure by using IBM PowerVC on your server. This flexibility helps ensure accessibility, whether you are migrating to hybrid cloud, fully embracing the cloud, or maintaining an on-premises setup. Regardless, IBM i Merlin CI/CD and IDE capabilities remain versatile and accessible.

IBM i VM provisioning with Ansible automation has the following capabilities:

- ▶ Supports PowerVC and Power Systems Virtual Server environments.
- ▶ Supports direct VM provisioning from templates, facilitating integration with CI/CD tools.
- ▶ Enables software installation on IBM i platforms, aligning with hybrid cloud strategies.

IBM i Merlin incorporates specialized Ansible playbooks for PowerVC and Power Systems Virtual Server environments. Although this approach is not the usual one due to often static IBM i logical partition (LPAR) structures, administrators can enable this function.

Complete the following steps:

1. Initiate the process by crafting a PowerVC template within the inventory. Figure A-6 shows a visual depiction of the Inventory interface, which facilitates this essential step in the workflow.

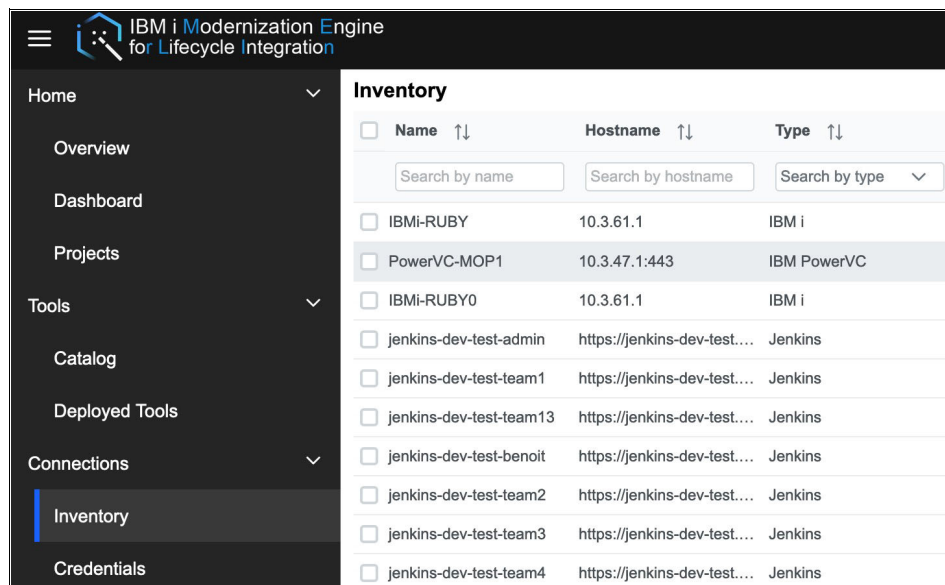


Figure A-6 Creating a PowerVC template in the Inventory

2. Configure the PowerVC credentials, enabling IBM i Merlin to securely communicate with the PowerVC instance. Provide the necessary authentication details to establish a connection. Figure A-7 on page 359 shows editing the Inventory.

**Edit Inventory**

\*Name:

Type:

\*Hostname:

\*Port:

Owner:

Description:

Figure A-7 Editing the Inventory to configure the PowerVC credentials

- Use the IBM i Merlin GUI to initiate VM provisioning, which is working with a IBM i Merlin Template. This process simplifies the creation of VMs, facilitating the deployment of IBM i instances on PowerVC or Power Systems Virtual Server. Follow the instructions that are shown in Figure A-8 and input the necessary details to customize your VM's configuration.

**Add Template**

**Basic Configuration**

\*Name:

\*Inventory:

\*Credential:

Type:

Owner:

Description:

**Virtual machine Configuration**

\*Virtual machine name:

\*Compute template:

\*Source image:

\*Primary network:

Run CL command:

Figure A-8 Adding a Template in the IBM i Merlin GUI for VM provisioning

- After completing the configuration, a dedicated IBM i Merlin menu becomes available within the GUI, offering access to the VM provisioning process. This feature transforms IBM i Merlin into a central hub that permits developers to provision PowerVC or Power Systems Virtual Server VMs that are tailored for modernization projects. New VMs integrate dynamically into the Inventory for use by IBM i Developer or CI/CD services. In Figure A-9, select **Provision** → **Deploy Virtual Machine**.

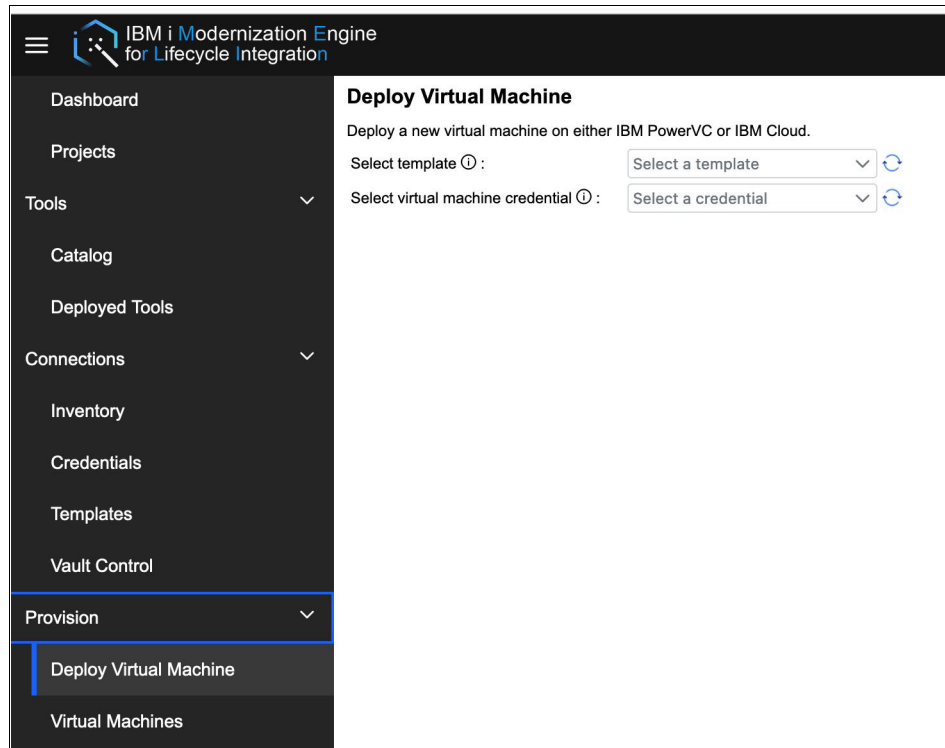


Figure A-9 Navigating the Deploy Virtual Machine menu in the IBM i Merlin GUI

**Note:** Similar steps apply to Power Systems Virtual Server provisioning, requiring an IBM Cloud API Key for credentials. For more information, see [Deploy IBM i server with PowerVC Template](#).

Within the IBM i Merlin pod, a set of Ansible playbooks and the Ansible engine facilitate internal setup, initialization of the IBM i Merlin-IBM i environment, and Power Systems Virtual Server or PowerVC provisioning.

**Note:** IBM i Merlin does not incorporate Terraform because Terraform is not used as an automation tool for provisioning IBM i VMs within IBM i Merlin. Instead, the automation tool is Ansible.

IBM i Merlin 1.0 introduces an Ansible Controller that is integrated into the IBM i Merlin engine. This controller orchestrates the following items:

- ▶ Internal playbooks for IBM i VM provisioning by using PowerVC or Power Systems Virtual Server through OpenStack and IBM Cloud modules. These playbooks include VM Provisioning and VM Destroy.
- ▶ A default set of six playbooks, which are referred to as "actions" in IBM i Merlin.



Administrators must perform these actions in a sequential manner to prepare the target IBM i LPAR for efficient management by IBM i Merlin. These actions pave the way for development with IBM i Merlin, build processes, and CI/CD practices.

- a. Enabling Ansible: Installs essential packages such as yum, Python, and Ansible on the IBM i server.
- b. Validating PTF Level: Verifies the Program Temporary Patch (PTF) level by using IBM i Merlin.
- c. Installing Certificates: Facilitates secure communication by installing the necessary certificates.
- d. Enabling IBM i developer: Enables the IBM i developer environment.
- e. Enabling a remote debugger: Permits administrators to enable the remote debugger feature.
- f. Enabling ARCAD environment: Installs ARCAD solutions on the target IBM i system.

These actions are shown in Figure A-10.

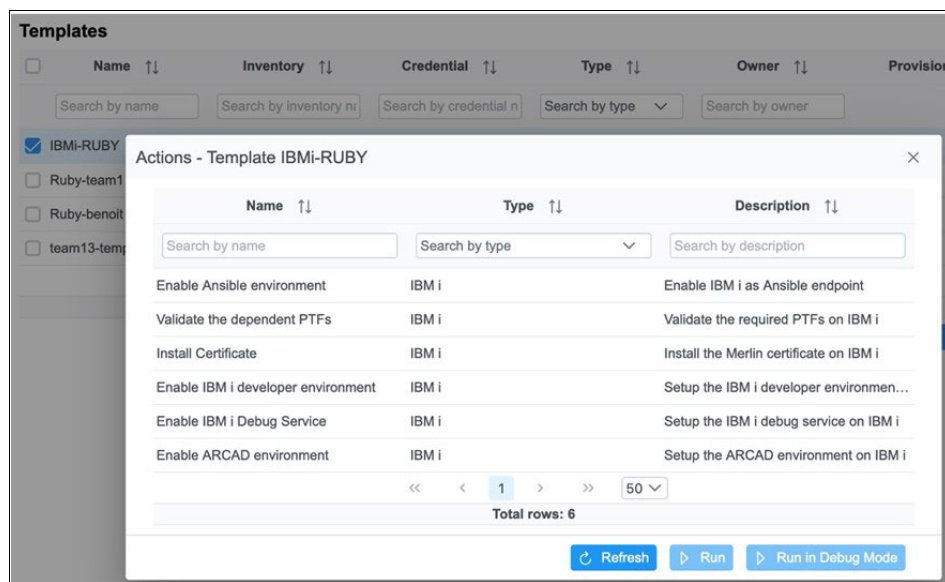


Figure A-10 Six actions on IBM i that are performed from IBM i Merlin by the administrator

**Note:** In future releases, based on the evolving product roadmap that is subject to potential changes, more playbooks will be introduced. These forthcoming playbooks will use tasks such as PTF management, Security and Compliance management, and more by using Ansible to effectively manage IBM i environments. However, such extra playbooks are not included in Version 1.0, reflecting the aim to keep IBM i Merlin supported for automation even for those users with limited Ansible knowledge or skills.

## The business demands for DevOps on IBM i

The path to modernization in the ever-evolving technological landscape calls for a comprehensive approach. Here are essential steps to revamp your development processes and embrace the principles of DevOps:

- ▶ Encourage a natural skills transfer by fostering a mindset shift and cultivating champions within your teams, helping ensure effective knowledge dissemination.
- ▶ Prioritize security by implementing new tools that meet critical requirements and elevate your overall security measures.
- ▶ Address complex application architectures by using specialized tools to re-factor the architectures, migrating toward more flexible and services-based structures, which enable updates and modernization.

### DevOps MVP architecture overview: Preceding IBM i Merlin

In the world of DevOps environments, the landscape preceding the advent of IBM i Merlin often featured intricate setups. Many clients initiated the process of setting up DevOps frameworks by using tools such as Jenkins, Ansible, and others. Such initiatives aligned well with the prevalent practices in their software development and deployment across different platforms.

However, for the IBM i ecosystem, these attempts resulted in a force-fit scenario because the unique requirements of the IBM i platform needed to be integrated within these existing frameworks. Figure A-11 illustrates the complexities and challenges that are faced by those developers striving to align their processes. This diagram showcases the DevOps landscape prevalent before IBM i Merlin, emphasizing the need for a more tailored solution.

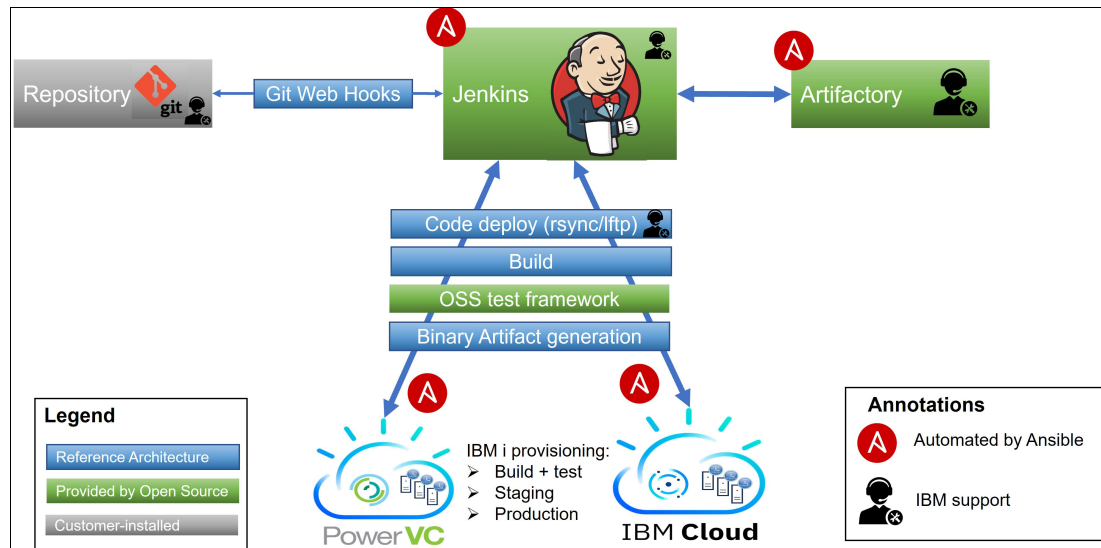


Figure A-11 DevOps MVP Architecture and integration diagram before IBM i Merlin

### Full cycle of the CI/CD process on IBM i with Ansible

This section explores an in-depth understanding of the DevOps pipeline process, commencing with CI/CD. We illustrate this process by using various tools and solutions, including Git, Ansible, and IBM PowerVC (OpenStack).

Figure A-12 on page 363 shows the architecture. The primary focus is on CI, although CD requires more workflows.

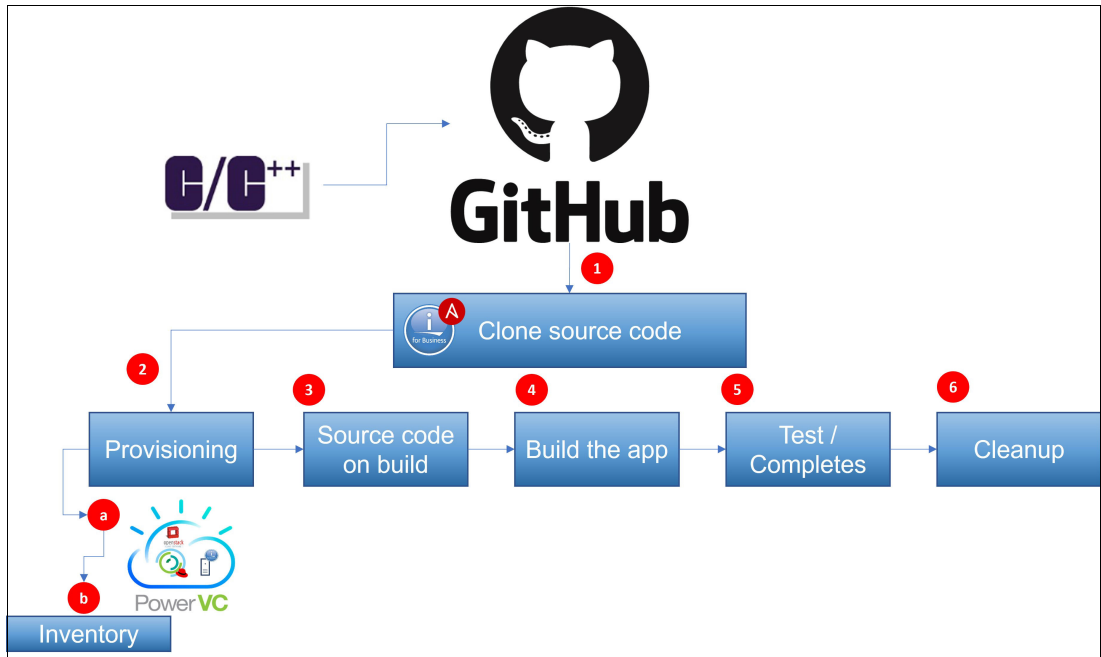


Figure A-12 End-to-end CI/CD process diagram for IBM i with Ansible

Examine the playbooks that are central to this use case. The set of playbooks, along with an inventory file and the Ansible configuration file, is outlined in Example A-1.

Example A-1 Set of playbooks to run full cycle of the CI/CD process on IBM i

```
-- add_build_system.yml
-- ansible.cfg
-- build.yml
-- cleanup.yml
-- git_clone.yml
-- hosts.ini
-- main.yml
-- post_build_actions.yml
-- provision_vars.yml
-- provision_vm.yml
-- put_code.yml
```

The core playbook for CI/CD is `main.yml`. Example A-2 offers clear insight into the upcoming run process.

Example A-2 Principal playbook for CI/CD: `main.yml`

```
---
- hosts: localhost
  vars:
    build_with_stmfs: true
    provision: true
    cleanup: true
  vars_prompt:
    - name: build_number
      prompt: "Enter a build number"
      private: no
```

```

    - name: git_repo_url
      prompt: "Enter a Git repo URL"
      private: no
    - name: git_branch
      prompt: "Enter a Git branch"
      private: no
collections:
  - ibm.power_ibmi
tasks:
  - set_fact:
      build_lib: "BUILD_{{ build_number }}"

  - set_fact:
      build_path: "/tmp/{{ build_lib }}"
      local_workspace: '~/workspace/{{ build_lib }}'

  - block:
      - name: Step 1 - clone source code from Git
        include: git_clone.yml

      - block:
          - name: include provision related vars if provision is true
            include_vars: provision_vars.yml

            - name: Step 2.1 - provision vm on demand
              include: provision_vm.yml
              when: provision

          - name: Step 2.2 - add build system to in-memory inventory
            include: add_build_system.yml

          - name: Step 3 - put source code to build machine
            include: put_code.yml

          - name: Step 4 - build your app on a build machine
            include: build.yml

          - name: Step 5 - run test and completes
            include: post_build_actions.yml
            delegate_to: "build_system"

  always:
    - name: Step 6 - cleanup on demand
      include: cleanup.yml
      when: cleanup
...

```

---

The steps that are outlined in `main.yml` call distinct YAML files:

1. Clone source code: When exploring the realm of CI/CD, the underlying objective remains consistent. Essentially, you initiate a pipeline with an input (often your source code) and an outcome is generated, which can be a packaged program. It is likely that the initial step in your pipeline predominantly involves cloning.

The subsequent YAML file, which is dedicated to this task, follows a sequence:

- a. If a local workspace exists on the system, it is removed.
- b. A new local workspace is created at the localhost.
- c. Clone a Git repository at the localhost, where the localhost functions as the IBM i control node in this context.

The prerequisites for this action include the repository URL, the previously created local workspace, and the designated Git branch.

The variables pertaining to this process are specified within the `main.yml` file. The YAML file that is dedicated to the cloning process is shown in Example A-3.

*Example A-3 Playbook for cloning the source code*

---

```
---
- name: remove if {{ local_workspace }} exists
  file:
    path: '{{ local_workspace }}'
    state: 'absent'

- name: create {{ local_workspace }}
  file:
    path: '{{ local_workspace }}'
    state: 'directory'
    mode: '0755'

- name: git clone from source repository
  git:
    repo: '{{ git_repo_url }}'
    dest: '{{ local_workspace }}'
    version: '{{ git_branch }}'
...
```

---

2. Provisioning: The provisioning phase emerges as a crucial cornerstone of the overall process. This segment encapsulates two distinct elements, each playing a significant role in the smooth configuration and deployment of the IBM i VM. These components can be delineated as follows:

- a. Provisioning variables: This YAML file serves as a repository for predefined variables that are used in provisioning the IBM i VM. This process uses PowerVC related information. Most of the variables within this context remain static, eliminating the need for further modifications.

A fresh user profile that is named “BUILDER” was created by cloud-init. The structure of this file is shown in Example A-4.

*Example A-4 Playbook repository for predefined variables for provisioning*

---

```
---
powervc_host: "192.168.1.101"
powervc_admin: "root"
powervc_admin_password: "abc123"
project: ibm-default
project_domain: Default
user_domain: Default
vm_name: "VM-{{ build_lib }}"
verify_cert: false
image_name_or_id: "IBMi_73"
nic_list: [{ 'net-name': 'Network1' }]
flavor_name_or_id: tiny
deploy_timeout: 300
deploy_userdata: |
  {% raw %}#!/bin/sh
  mkdir /home/BUILDER
  system "CRTUSRPRF USRPRF(BUILDER) PASSWORD(abc123) USRCLS(*SECOFR)
  ASTLVL(*SYSVAL) TEXT('Ansible CICD build') HOMEDIR('/home/BUILDER')"
  system "chgtcpsvr svrspcval(*sshd) autostart(*yes)"
  system "strtcpsvr *sshd"
  {% endraw %}
...

```

---

- b. Provision virtual machine: This YAML file includes a PowerVC host into the in-memory inventory, effectively supplanting the manual approach of editing the inventory file for host addition. While provisioning, the `os_server` compute instance from OpenStack is employed. Post-provisioning, an extra task runs to sift through the output, revealing the IP address of the freshly provisioned IBM i VM, which is shown in Example A-5.

*Example A-5 Playbook that introduces a PowerVC host into the in-memory inventory*

---

```
---
# Add PowerVC host to in-memory inventory
- name: Add PowerVC host {{ powervc_host }} to the Ansible in-memory inventory
  add_host:
    name: 'powervc'
    ansible_user: '{{ powervc_admin }}'
    ansible_ssh_pass: '{{ powervc_admin_password }}'
    ansible_ssh_extra_args: -o StrictHostKeyChecking=no
    ansible_python_interpreter: /usr/bin/python3
    ansible_ssh_host: '{{ powervc_host }}'
  no_log: true

# New vm information from OpenStack
- name: Deploy a new VM
  os_server:
    auth:
      auth_url: https://{{ ansible_ssh_host }}:5000/v3
      username: '{{ ansible_ssh_user }}'
      password: '{{ ansible_ssh_pass }}'
      project_name: '{{ project }}'
      project_domain_name: '{{ project_domain }}'

```

```

    user_domain_name: '{{ user_domain }}'
    name: '{{ vm_name }}'
    image: '{{ image_name_or_id }}'
    flavor: '{{ flavor_name_or_id }}'
    verify: '{{ verify_cert }}'
    nics: '{{ nic_list }}'
    timeout: '{{ deploy_timeout }}'
    userdata: '{{ deploy_userdata }}'
    register: vm_info
    delegate_to: 'powervc'

- name: New IBM i VM's IP
  debug:
    msg: "{{ vm_info.server.accessIPv4 }}"
  ...

```

- 
3. Add build system: This YAML file introduces the IBM i VM that is deployed in the in-memory inventory. The inventory is populated with values by using `set_fact`. The IP address of the IBM i VM, which is obtained from the register during deployment (`vm_info.server.accessIPv4`), is a key inclusion.

Also, values from variables such as `ansible_ssh_user` and `ansible_ssh_pass`, which are defined in the `hosts.ini` file, are integrated. The `known_hosts` module plays a role in adding or removing the host key (Secure Shell (SSH)) for the deployed IBM i VM. This key is essential for the control node to manage the new VM through Ansible. The term “non-patched” refers to a new VM, which requires verification of the PGM.

Another crucial module that is named `wait_for_connection` is part of this process. This module monitors the new VM's status until it successfully establishes an SSH connection. The managed node requires Python 3 and its associated packages. These prerequisites are installed by using the `raw` module. The structure of the YAML file is shown in Example A-6.

*Example A-6 Playbook that introduces the IBM i VM deployed in-memory inventory*

---

```

---
- block:
  - name: set_fact for non-patched build environment
    set_fact:
      build_system_ip: "{{ vm_info.server.accessIPv4 }}"
      build_system_user: "{{ hostvars["non-patched"]["ansible_ssh_user"] }}"
      build_system_pass: "{{ hostvars["non-patched"]["ansible_ssh_pass"] }}"

  - name: remove existing entry for vm in case ssh header change occurs.
    known_hosts:
      name: "{{ build_system_ip }}"
      path: ~/.ssh/known_hosts
      state: absent

  - name: add vm-{{ build_lib }} to ansible in-memory inventory
    add_host:
      name: build_system
      ansible_ssh_host: '{{ build_system_ip }}'
      ansible_user: '{{ build_system_user }}'
      ansible_ssh_pass: '{{ build_system_pass }}'
      groups: build_systems
      ansible_ssh_extra_args: -o StrictHostKeyChecking=no
      ansible_python_interpreter: /Q0pensys/pkgs/bin/python3

```

```

- name: wait until vm-{{ build_lib }} is up and ssh ready
  wait_for_connection:
    sleep: 10
    timeout: 1800
    delegate_to: "build_system"

- name: install python3 on VM-{{ build_lib }}
  raw: /QOpensys/pkgs/bin/yum install -y python3 python3-itoolkkit python3-ibm_db
  delegate_to: "build_system"
when: provision
...

```

- 
4. Put code: This YAML file orchestrates a sequence of tasks that are essential for code deployment. Initially, the `ibmi_cl_command` module establishes a library in the new VM. This operation is delegated to the IBM i controller node. Next, login credentials for authorized access to the newly created IBM i VM are placed into the `.netrc` file. This file is in the home directory of the IBM i controller node.

A set of tasks occur within a `block` structure for path creation. The `ansible.builtin.file` module is employed in this process to enable the creation of directories, which is guided by the specified variables from `main.yml`. Notably, this approach encompasses subdirectories as required, which helps ensure a comprehensive directory structure.

The next task transfers the 'C' program from the IBM i control node to the new IBM i VM. To facilitate this transfer and eliminate interactive prompt passwords, use the `sshpass` utility. Example A-7 shows the structure of the YAML file.

---

*Example A-7 Playbook for orchestrating the tasks for code deployment*

---

```

---
- name: Create {{ build_lib }} on {{ build_system_ip }}
  ibmi_cl_command:
    cmd: CRTLIB {{ build_lib }}
    delegate_to: "build_system"

- name: "check if ~/.netrc contains IBM i target login"
  lineinfile:
    name: ~/.netrc
    line: "machine {{build_system_ip}} login {{build_system_user}} password
    {{build_system_pass}}"
    state: present
    check_mode: false

- block:
  - name: Create {{ build_path }} on remote IBM i
    ansible.builtin.file:
      path: "{{ build_path }}"
      state: "directory"
      delegate_to: "build_system"

  - name: combine transfer_command
    set_fact:
      transfer_command: "scp {{ local_workspace }}/sendMsg.c
      {{build_system_user}}@{{build_system_ip}}:{{ build_path }}/sendMsg.c"

  - name: put STMFs to remote IBM i
    shell:
      cmd: 'sshpass -p "{{ build_system_pass }}" {{ transfer_command }}'

```



```
when: build_with_stmfs
```

```
...
```

- 
5. **Build:** This YAML file orchestrates the running of crucial tasks within the build process. The `ibmi_cl_command` modules start the Create Bound C++ Program (CRTBNDCPP) command, which initiates the Integrated Language Environment (ILE) C++ compiler. This operation uses specific variables that are defined in `main.yml` for parameterizing the IBM i command. The `<build_lib>` and `<build_path>` variables are among the variables that are employed.

The source stream file (SRCSTMF) accommodates the program's source code. This code is initially cloned from the Git repository, and then transferred to the newly created IBM i VM. The program source file that is named `sendMsg.c` is involved in this process. On compilation, an ILE C++ program object that is named `SENDMSG` is generated.

Example A-8 shows the structure of the YAML file to provide insight into the build process.

*Example A-8 Playbook to orchestrate the running of tasks within the build process*

---

```
---
- block:
  - name: call CL command to build application
    ibm.power_ibmi.ibm_i_cl_command:
      cmd: CRTBNDCPP PGM({{ build_lib }}/SENDMSG) SRCSTMF('{{ build_path
}}/sendMsg.c')
    when: build_with_stmfs
    delegate_to: 'build_system'
...

```

- 
6. **Post-build actions:** The focus shifts to the running of essential tasks after the build process. The pivotal task is initiating the `SENDMSG` program, which is followed by the registration of the output task. This outcome is systematically filtered to present the output that results from the program invocation.

The `when` directive evaluates a predefined condition, which is denoted as `true` within `main.yml`. This conditional assessment serves as the enabling factor for the running of the task program, which helps ensure its activation in the appropriate scenario.

The structure and sequence of tasks pertaining to post-build actions are outlined in Example A-9.

*Example A-9 Playbook for running built programs with stream files*

---

```
---
- name: run PGM built with STMFs
  ibm.power_ibmi.ibm_i_cl_command:
    cmd: CALL {{ build_lib }}/SENDMSG
    Job log: true
  register: callpgm
  when: build_with_stmfs
- name: PGM output
  debug:
    var: callpgm.job_log[0].MESSAGE_TEXT
...

```

7. Cleanup: In this YAML file (shown in Example A-10), the playbook removes the local workspace and directories from the newly created IBM i VM. Also, the playbook deletes the IBM i VM, which tests the program. The **pause** module, coupled with a **prompt**, helps ensure that the cleanup tasks proceed only when you press the Enter key.

*Example A-10 Playbook for cleanup and virtual machine deletion*

---

```
---
- pause:
  prompt: Confirm you want to cleanup! Press enter to continue

- name: remove "{{ local_workspace }}"
  file:
    path: '{{ local_workspace }}'
    state: 'absent'
  ignore_errors: true

- name: Destroy VM when provision
  os_server:
    auth:
      auth_url: https://{{ ansible_ssh_host }}:5000/v3
      username: '{{ ansible_ssh_user }}'
      password: '{{ ansible_ssh_pass }}'
      project_name: '{{ project }}'
      project_domain_name: '{{ project_domain }}'
      user_domain_name: '{{ user_domain }}'
    name: '{{ vm_name }}'
    state: 'absent'
  delegate_to: 'powervc'
  when: provision
...
```

---

**Note:** Before IBM i Merlin, DevOps environments often involved intricate setups that used tools such as Jenkins and Ansible. Although they were effective for various platforms, integrating IBM i requirements posed unique challenges. The DevOps MVP architecture overview, which has steps such as cloning source code, provisioning, and more, highlights the complexities that are faced in harmonizing processes. This retrospective underscores the IBM i Merlin role in offering a more tailored and efficient IBM i DevOps solution.

## IBM i Merlin DevOps CI/CD view

The foundation of the IBM i Merlin DevOps CI/CD view stems from a visionary concept. By integrating ARCAD into the IBM i Merlin product, it offers a unified solution that is steered by the IBM i Merlin UI and supported by ARCAD and IBM. This collaborative approach harnesses the collective expertise of both entities, resulting in a cohesive and comprehensive offering.

The focal point of the view revolves around DevOps, which drives the CI/CD pipeline. Within this domain, a crafted suite of tools has emerged that is based on Git and Jenkins. These tools are designed to strengthen the IBM i platform by facilitating a dynamic approach to continuous development, which involves the automated compilation of code segments that are extracted from RPG or COBOL applications, and then deployed to diverse IBM i endpoints. The orchestration of these tasks is expertly managed by Jenkins. For a visual depiction, see Figure A-13 on page 371.

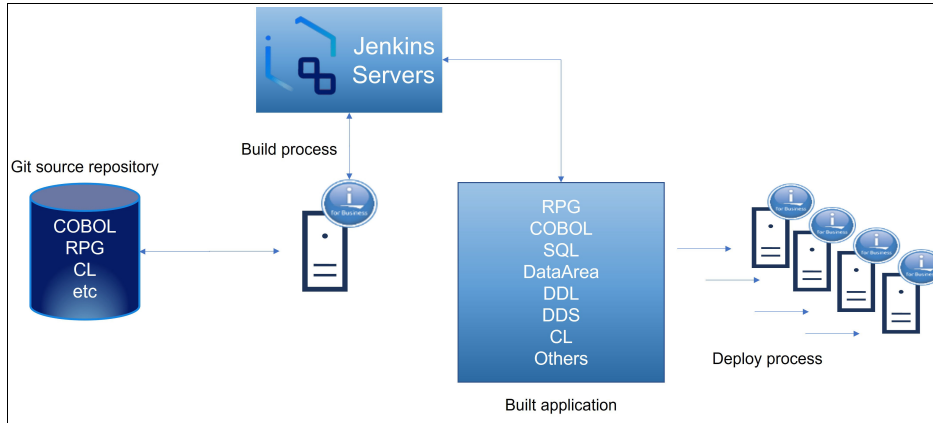


Figure A-13 Enhancing IBM i platform with Git and Jenkins in the DevOps CI/CD pipeline

Furthermore, the collaborative partnership between ARCAD and IBM yielded a deep understanding of the specific needs and intricacies of the IBM i platform. This knowledge has been instrumental in fine-tuning the integration of ARCAD's solutions with IBM i Merlin, helping ensure a harmonious alignment with IBM i requirements. The robustness of this collaboration is exemplified by ARCAD's suite of plug-ins that interact with IBM i Merlin's capabilities, facilitating a well-integrated and efficient development experience. The resulting synergy between ARCAD's expertise and IBM i Merlin's capabilities empowers organizations to achieve optimized DevOps practices and realize the full potential of their IBM i investments.

Figure A-14 shows a visual representation of this enriching collaboration.

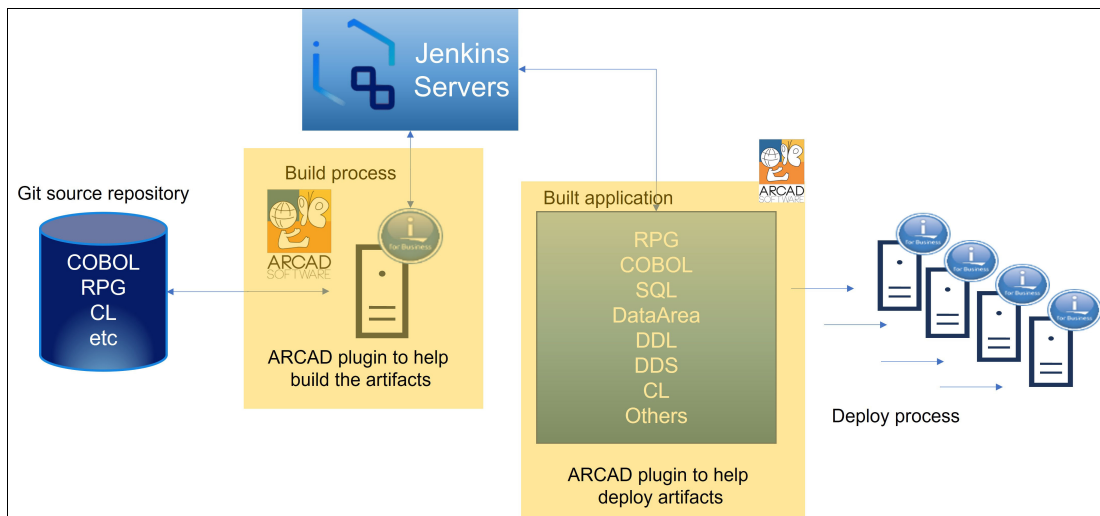


Figure A-14 Enriching the DevOps landscape: ARCAD integration with IBM i Merlin

### **IBM i Merlin architecture on Red Hat OpenShift Container Platform**

In the context of the IBM i Merlin architecture on the Red Hat OpenShift Container Platform, an important addition enters the picture. This new facet is the role of an Red Hat OpenShift environment manager. The goal is to facilitate a straightforward learning curve for those developers who lack an existing Red Hat OpenShift administrator within their setup. The objective is to impart essential knowledge about setting up an Red Hat OpenShift environment and integrating IBM i Merlin into it without any hindrance. This process simplifies the installation of IBM i Merlin within the Red Hat OpenShift environment, much like installing software on a developer's PC. However, the distinction lies in the installation occurring on the designated server platform, allowing team members to access it conveniently through their web browsers.

Figure A-15 shows how a Red Hat OpenShift administrator assumes the responsibility of overseeing all container-based applications that are operating on the Red Hat OpenShift platform.

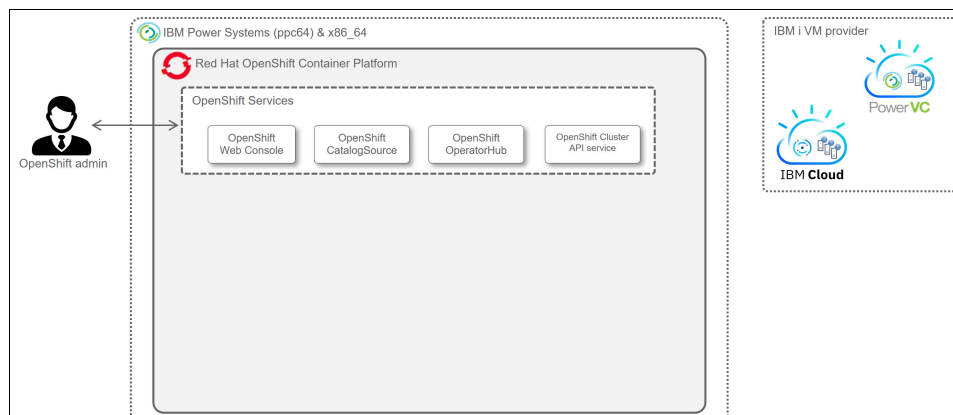


Figure A-15 Red Hat OpenShift administrator role in an IBM i Merlin integration

This role encompasses the following operations that are related to IBM i Merlin:

- ▶ Using the Red Hat OpenShift CLI or GUI console to gain access.
- ▶ Running the initial installation and deployment of the IBM i Merlin platform.
- ▶ Reviewing and managing objects, configurations, and settings that are specific to IBM i Merlin.
- ▶ Conducting platform upgrades for IBM i Merlin.
- ▶ Gathering various tiers of logs for IBM service evaluation.

**Note:** The Red Hat OpenShift environment can be on an IBM Power server or any location that is compatible with current Red Hat OpenShift implementations. Also, Red Hat OpenShift can be hosted within a cloud instance, such as IBM Cloud (IBM Power Systems Virtual Servers), or within any cloud platform that accommodates Red Hat OpenShift environments. Clients who have workloads functioning in the cloud can extend their operations by integrating IBM i Merlin into an Red Hat OpenShift environment within the cloud.

IBM i Merlin is designed for Red Hat OpenShift containers, and is applicable to both IBM Power (ppc64) and x86 architectures.

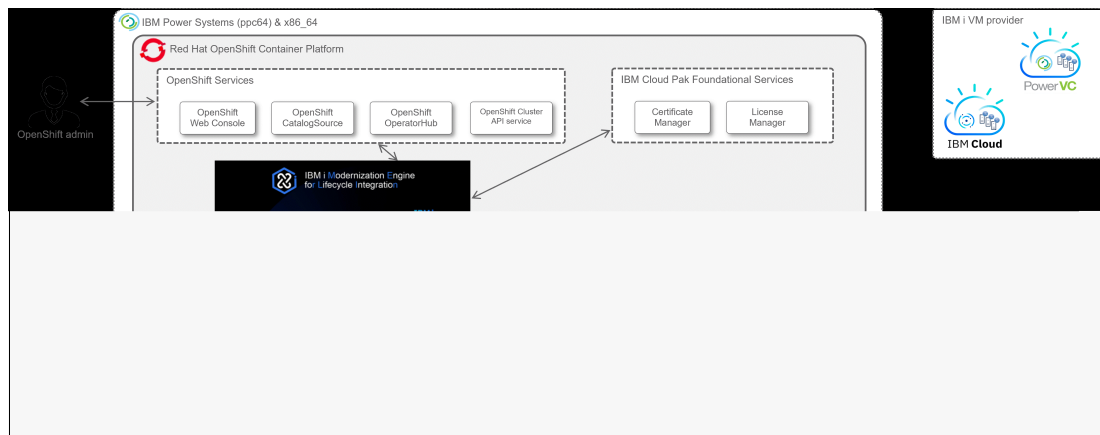
### ***The IBM i Merlin platform in the IBM i Merlin architecture***

IBM i Merlin administrators are tasked with helping ensure the efficient integration and connectivity of all applications within the IBM i Merlin platform. Their responsibilities encompass setting up the Git repositories and establishing the connections between components. A foundational understanding of concepts such as Git and Jenkins helps direct resources to the libraries and repositories.

**Note:** While some familiarity with Git and Jenkins is essential, in-depth knowledge of Linux or Red Hat OpenShift is not a prerequisite for the role.

In smaller environments, these responsibilities can be undertaken by a single individual who possesses the skills to initiate Red Hat OpenShift, facilitate its installation, and configure IBM i Merlin for operational use.

Figure A-16 shows the role and responsibilities of an IBM i Merlin administrator.



*Figure A-16 Role and responsibilities of an IBM i Merlin administrator*

In Figure A-16, a IBM i Merlin administrator assumes a pivotal role, wielding direct access to the IBM i Merlin platform GUI while orchestrating a spectrum of activities that are tailored for IBM i users. The scope of responsibilities requires crucial operations to help ensure the platform's efficient functioning for IBM i users:

- ▶ Installing and deploying IBM i Merlin tools, which encompass components such as the IDE, CI/CD functions, and more.
- ▶ Proficiently managing user accounts within the IBM i Merlin ecosystem.
- ▶ Helping ensure the security and confidentiality of sensitive information that is entrusted to the platform.
- ▶ Exerting control over authorities and permissions to establish a defined hierarchy of access.
- ▶ Diligently monitoring resource consumption and utilization to optimize the platform's efficiency.
- ▶ Collaborating closely with Red Hat OpenShift administrators to promptly address and resolve potential issues to nurture a synergistic partnership for the platform's overall integrity and reliability.

## IBM i Merlin platform and tools architecture

In the context of the IBM i Merlin platform and tools architecture, a distinct focus emerges. The objective is to empower developers, enabling them to engage in their development tasks by using intuitive, browser-based interfaces. This anticipated progression involves a certain learning curve, which is vital for mastering the installation of the components that depend on the operational flow. Beyond this initiation, the journey is characterized by the accessibility of browser-driven functions. These functions play a pivotal role in orchestrating fundamental CI/CD processes to enhance the efficiency of the development experience.

Figure A-17 illustrates how to enable developers by using the IBM i Merlin platform and tools.

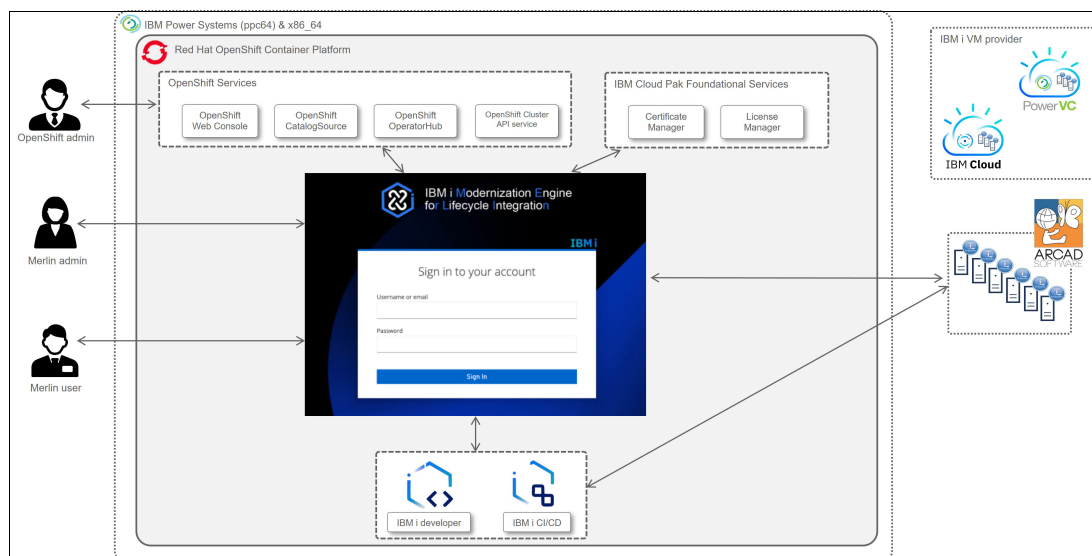


Figure A-17 Empowering developers with IBM i Merlin platform and tools

In Figure A-17, a IBM i Merlin user has direct access to the IBM i Merlin platform and tools. This user engages with IBM i Merlin to achieve the following objectives:

- ▶ Access and use the deployed IDE, CI/CD, and other functions.
- ▶ Manage inventory, user profiles, and credentials for targeted systems.
- ▶ Oversee the lifecycle of the IBM i Merlin tools, subject to the prerequisite authority.
- ▶ Create RESTful services on designated IBM i systems.

## IBM i Merlin tools for IBM i CI/CD

Drawing on ARCAD's expertise in IBM i DevOps, the approach to deployment offers versatility to suit different scenarios. You can use your existing Jenkins server or use standard setup, where IBM establishes a Jenkins server for you and integrates it effectively with ARCAD components. This integration simplifies your CI/CD workflow and enhances efficiency. Importantly, IBM i Merlin provides a GUI that is tailored for IBM i to help ensure a smooth experience.

Figure A-18 on page 375 show a visual representation of these capabilities.

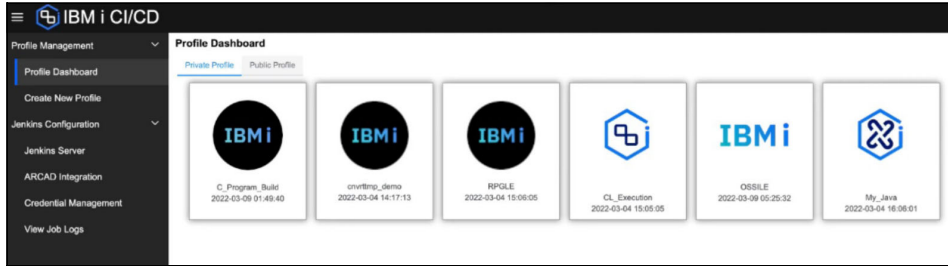


Figure A-18 IBM i CI/CD GUI with IBM i Merlin tools

Key features:

- ▶ Simplified Jenkins complexity: In IBM i Merlin, Jenkins is transparent, so you can concentrate on your IBM i CI/CD processes.
- ▶ Choice of deployment: You can choose to deploy without a Jenkins server, use your own Jenkins instance, or use the provided Jenkins server that is integrated with ARCAD plug-ins.
- ▶ Flexible profile management: Private and public profiles offer a robust means to generate Jenkins pipelines dynamically. These profiles can be easily shared among IBM i Merlin users to promote collaboration and consistent practices.

## IBM i Merlin for IBM i developers

In the context of code development, a natural development environment is crucial. Historically, IBM i featured PDM on the 5250, which is ingrained in the IBM i development landscape. However, PDM in the contemporary landscape is less normalized. The IBM Rational Developer product remains effective for numerous users.

Figure A-19 shows a representation of PDM.

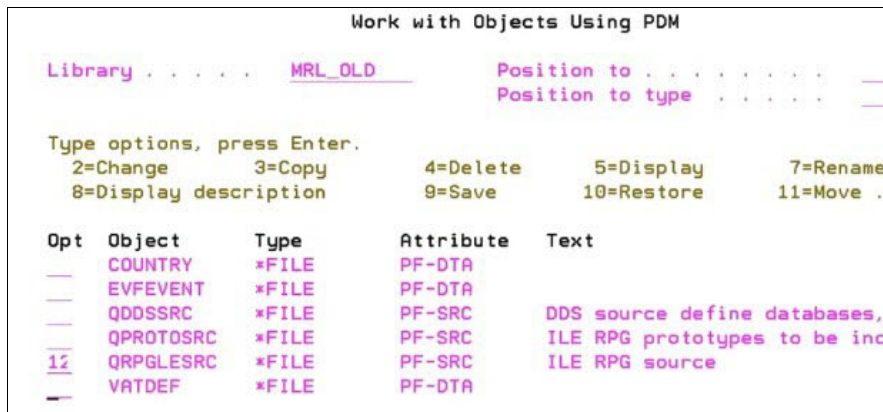


Figure A-19 Work with objects by using PDM on the 5250 interface

The primary aim is to help IBM i customers seeking to adopt Git as their source control repository while embracing a modern, browser-based development arena. Thus, IBM crafted a code-ready workspace by harnessing Eclipse Theia and Che along with an array of code plug-ins. These components converge to deliver genuine code comprehension, comprehensive formatting, and a deep understanding of languages such as RPG, COBOL, and other native ILE types.

ARCAD tools have played a pivotal role in development pursuits over the years. Features such as RPG conversion to modern free format and immediate access to an impact analysis tool are now essential aspects.

Figure A-20 shows the IDE, where you set up your development environment, incorporate rich editing capabilities, and build and compile your project. Designed for contemporary developers, the offering includes natural Git integration, intelligent build functions, self-contained projects, code comprehension, and integrated impact analysis.

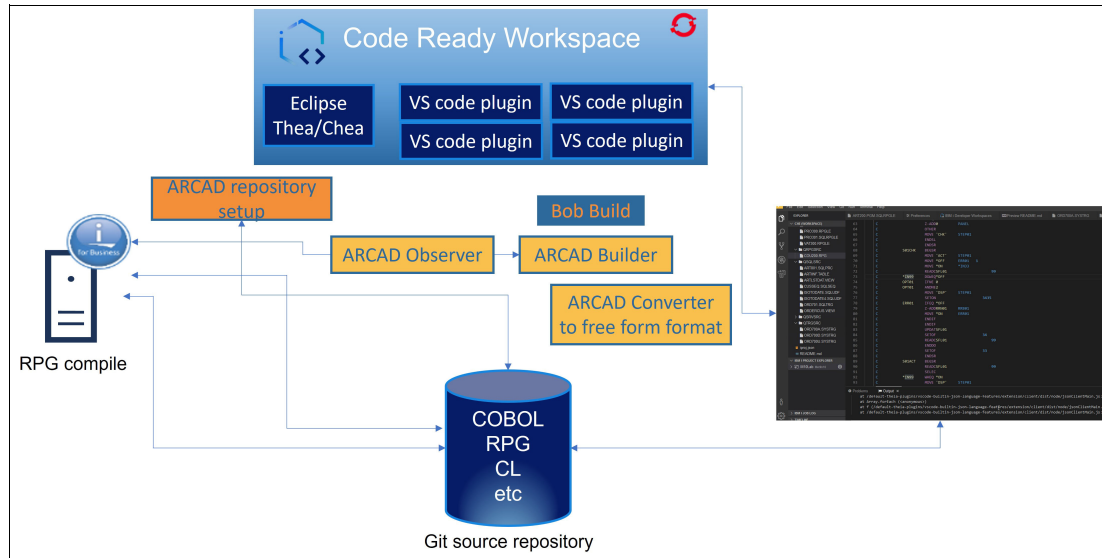


Figure A-20 Comprehensive development workflow with integrated tools

**Note:** Build on Build (BOB) is a tool that provided by ARCAD that facilitates the automated compilation and build process for RPG and COBOL applications on the IBM i platform. Developers can use it to initiate the build process directly from a visual board or interface to help streamline the development workflow and enhance efficiency. The BOB tool integrates with DevOps practices and CI processes for faster and more automated application builds.

## Coexisting advancements: IBM Rational Developer for i and IBM i Merlin in IBM i development

IBM i Merlin introduces a fresh approach to code development and modernization that coexists with IBM Rational Developer for i. Rather than replacing IBM Rational Developer for i, IBM i Merlin provides an alternative option for developers to choose between workstation-based development with IBM Rational Developer for i or the browser-based, container-oriented environment of IBM i Merlin.

IBM Rational Developer for i caters to creating and updating native ILE applications on IBM i, and IBM i Merlin uses a holistic CI/CD ecosystem that is based on Jenkins. It serves as more than just an IDE by offering a comprehensive suite of tools and plug-ins to facilitate modern development practices. IBM i Merlin equips developers with features such as Patched to Free conversion, integration with Git-based source control, and real-time application impact analysis. Also, it integrates with automated build and deployment pipelines, streamlining the entire development lifecycle.



A key distinction lies in the modernization capabilities of IBM i Merlin. Code that is crafted within IBM i Merlin can still be modified by using IBM Rational Developer for i. Although the Source Entry Utility (SEU) can also be used for further code modification, IBM i Merlin support for the latest RPG versions emphasizes a shift toward contemporary coding approaches, encouraging developers to permit newer paradigms for enhanced efficiency and sustainability.

### IBM i Merlin and ARCAD: Tools for enhanced development

IBM i Merlin has undergone significant enhancements that were achieved by integrating existing products and introducing novel elements. One example of this evolution is the builder facet, which is now enriched with a new web server and CLI. This integration includes various archive tools that are bundled with the core IBM i Merlin product. The core of this integration is the metadata repository, prominently featured in the lower right of Figure A-21. This repository facilitates shared access for critical components such as Builder, Transformer RPG, and Observer. It connects these products, enabling real-time awareness of any changes or additions to objects.

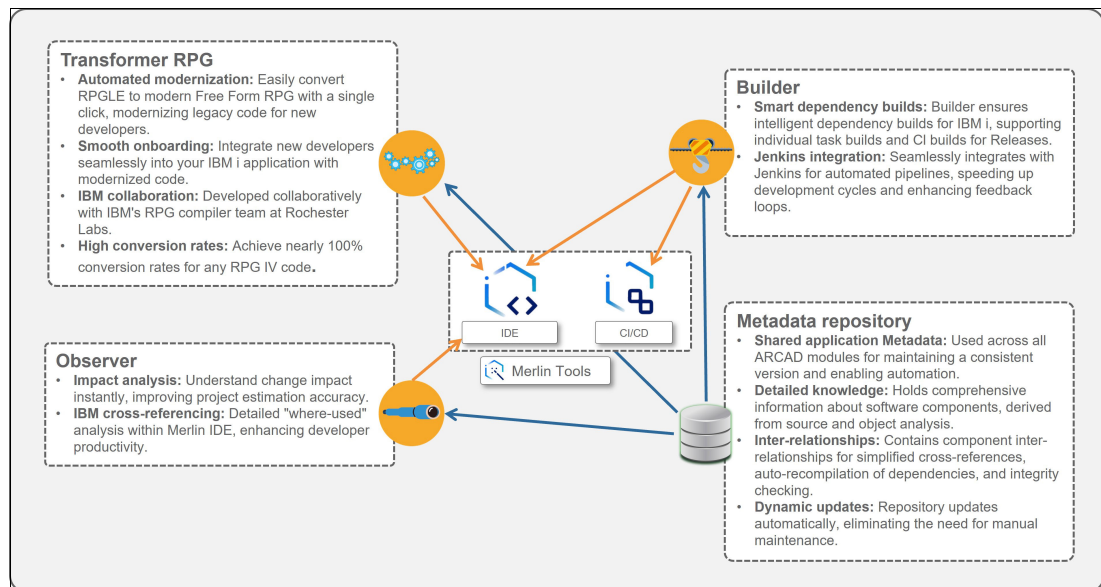


Figure A-21 IBM i Merlin and ARCAD: Tools for enhanced development

This robust integration extends its benefits to both the IDE and the CI/CD process. Users can initiate actions such as impact analysis directly from the IDE, and in the CI/CD pipeline objects are automatically built along with their dependencies because of the consistently maintained metadata repository. This automated process eliminates the necessity of manually managing makefiles, which is a task that can become unwieldy in enterprise-level settings. The Transformer RPG component serves as the initial step in the modernization journey. It enables migrating code to a fully free-format RPG before engaging the broader range of modern tools and coding capabilities that are offered by IBM i Merlin.

This intricate integration and comprehensive enhancement underscore the IBM i Merlin pivotal role as a potent modernization engine that is focused on lifecycle integration.

## Developer IBM i Merlin

The Developer IBM i Merlin environment provides a range of capabilities to enhance the development process:

- ▶ **Connections:** Includes features such as inventory management, credentials management, and template setup, which enable efficient access to the resources that are needed for development tasks.
- ▶ **Tools:** Developer IBM i Merlin offers a suite of tools, which includes tools that are deployed and configured for specific tasks. The IBM i Developer tool provides seamless integration to developers. With this tool, you can perform actions such as right-clicking to run applications, which streamline the development workflow.
- ▶ **Create a workspace:** This function establishes a dedicated workspace that is tailored to a developer's requirements to help ensure an organized and efficient development experience.

Figure A-22 shows the initial workspace within IBM i Developer, which is the starting point for developers that use IBM i Merlin.

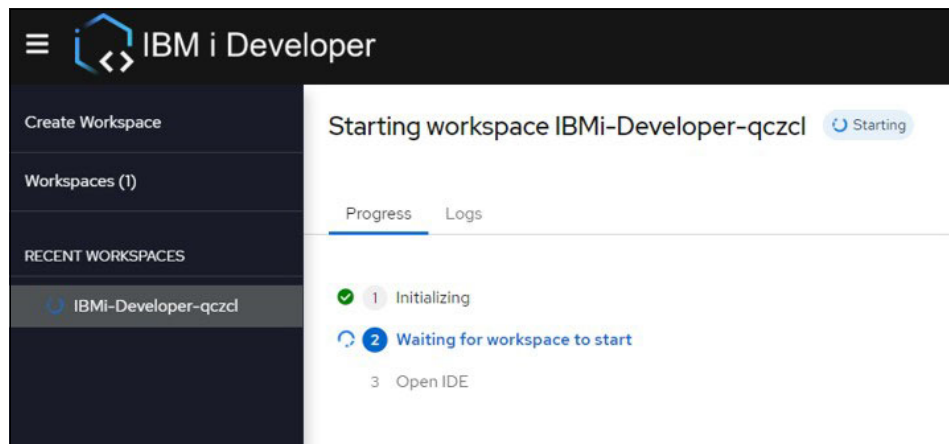


Figure A-22 Starting workspace in IBM i Developer

## Development flow

Here are the key principles that define the IBM i Merlin development flow:

- ▶ **Inspired by GitFlow:** The development flow model is inspired by the GitFlow methodology, which is an established branching strategy. This approach provides a structured framework for managing code changes, releases, and collaboration among developers.
- ▶ **Adaptable:** The development flow is designed to be adaptable and accommodate various project requirements and team dynamics. It can be tailored to the specific needs of the development team.
- ▶ **Master and development with no direct changes:** The primary development branches, "Master" and "Development," cannot be directly changed. Instead, developers work on feature branches or other specialized branches to help ensure that the main development branches remain stable and reliable.

- ▶ Other branches: The development flow includes several types of branches that serve distinct purposes:
  - *Feature branches* are created for developing new features or functions. These branches enable developers to work on isolated changes without affecting the main codebase.
  - *Release branches* are used to prepare the codebase for a new release. They are ideal for bug fixes, last-minute adjustments, and testing before a release.
  - *HotPatch branches* are created to address critical issues in the production environment. They enable swift fixes without interrupting ongoing development efforts.
- ▶ Branch for ARCAD version: Each branch that is created within the development flow is accompanied by an associated ARCAD version. This version management helps ensure proper tracking and integration of changes to provide clear visibility into the status and progress of development activities.

**Note:** Git, a distributed VCS, offers several compelling advantages for SCM:

- ▶ Line-level visibility of changes: Unlike traditional change management systems, Git provides a granular view of changes at the line level so that developers can precisely track modifications.
- ▶ Enhanced management of concurrent development: Git's decentralized nature enables multiple developers to work on different branches simultaneously, which facilitates smoother collaboration and concurrent development efforts.
- ▶ Explicit merges: When two changes are merged into the same codebase, Git makes this process explicit to help ensure that changes are intentionally combined, which reduces the risk of accidental conflicts.
- ▶ Controlled commits: Git's commit process includes conflict checks so that developers can review and manage potential conflicts before finalizing changes, which enhance code quality and reduces integration challenges.
- ▶ Offline usage: Git's offline capabilities enable developers to track local changes even when disconnected from a network. This flexibility supports productivity in various work environments.
- ▶ Incredible traceability: Git's version control offers unparalleled traceability so that you can track the history of changes, contributors, and decisions made throughout the development lifecycle.

## Managing IBM i source with Git and ARCAD

Effectively managing your IBM i source code is crucial for a development process. By combining the power of Git version control and ARCAD capabilities, you can optimize your source management workflow. The following steps outline the process of handling IBM i source code within this collaborative environment:

1. Create an empty Git repository: Begin by creating an empty Git repository to be the foundation for your SCM.
2. Configure your application in ARCAD: Configure your application within ARCAD by specifying components such as Mxx\_DTA, Mxx\_OBJ, and Mxx\_SRC. Link these components to the Git repository for seamless integration.

3. Generate source for objects: Use the GENSCMSRC command to generate source code specifically for objects. This process facilitates the progression of source code transition into your infrastructure.
4. Load the Git repository with your source: Use the LODSCMREP command to populate the Git repository with the generated source code. This step helps ensure that your code is effectively managed within the VCS, which enhances collaboration and traceability.

## IBM i Merlin preferences

Within IBM i Merlin, user preferences are designed to enhance the development experience. These preferences are tailored to integrate with ARCAD tools to help ensure a cohesive workflow.

1. Builder Port 5252: This preference configures the communication port for Builder to help ensure interaction between components.
2. IBM i Developer:
  - Build settings: The preferences for IBM i Developer encompass a range of build settings, which include options that are related to BOB.
  - Formatting options are available so that developers can tailor their development environment to their coding style and preferences.
3. Customizable color scheme: IBM i Merlin can modify the color scheme. Developers can use this customization feature to create a coding environment that is visually conducive to their individual needs.

## Git integration in IBM i Merlin

Integrating Git functions within the IBM i Merlin workspace brings enhanced efficiency and collaboration to your development process. The integration of the `git` command enables effective version control and access to various collaborative tools. The following points outline the key steps and benefits of this integration:

1. Press the F1 key to access a comprehensive list of Git commands.
2. Initiate the process by running `git clone` on the CLI of the IBM i Merlin workspace, as shown in Figure A-23.

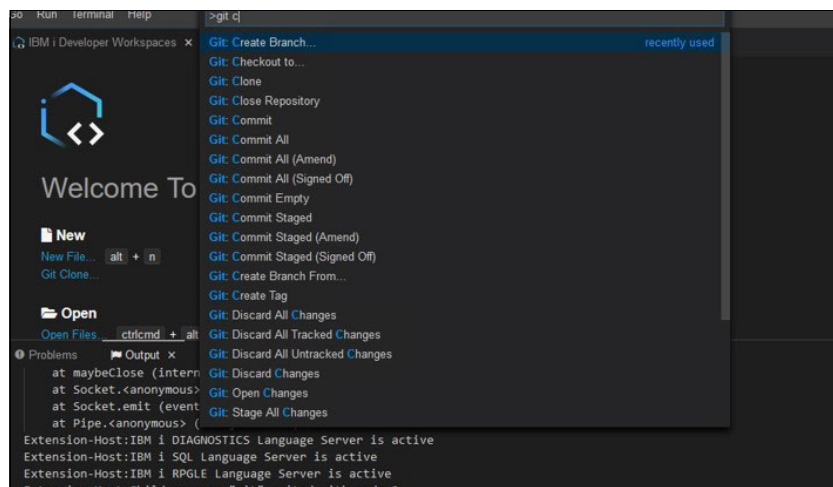


Figure A-23 Git clone command in IBM i Merlin workspace

3. Provide the SSH URL of the Git repository that you intend to clone to establish the connection between IBM i Merlin and the Git repository.

On successful completion of the clone process, your source code becomes visible and accessible within the IBM i Merlin workspace. This integration streamlines version control and SCM, which enhances your development workflow.

4. To create a branch, go to the “Feature/xxxx” section, where the mapping between Git and ARCAD, which is labeled as `awrkvertyp`, is defined. as shown in Figure A-24.

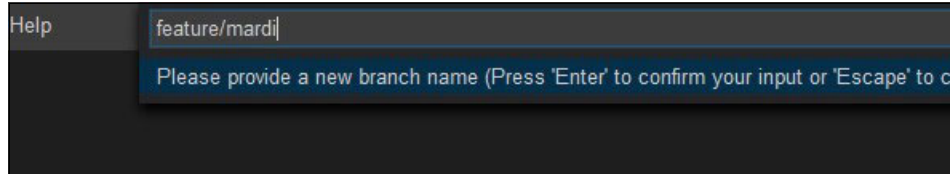


Figure A-24 Creating a branch in IBM i Merlin: Mapping Git and ARCAD

5. Press F1 and select `git create branch`, as shown in Figure A-25.

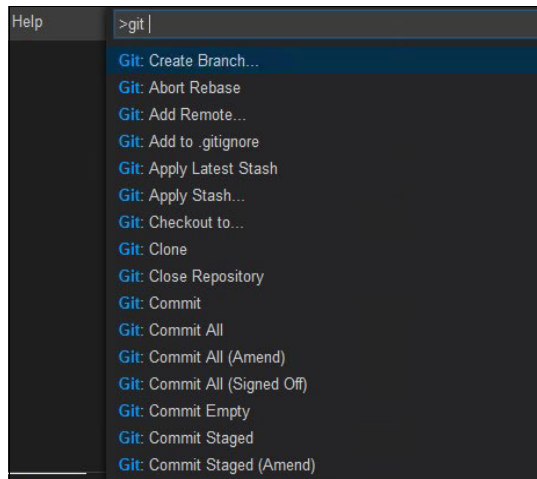


Figure A-25 Creating a branch in IBM i Merlin

6. In the lower left, click `master` to proceed with the branch creation process, as shown in Figure A-26.

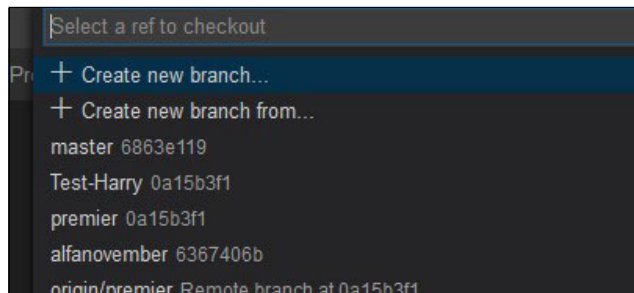


Figure A-26 Selecting master to begin branch creation

7. After changing your local repository, use the push command to upload your committed changes to the remote Git repository. This action synchronizes the changes that you made on your local machine with the online repository to help ensure that other team members can access your updates.

As a result of pushing your changes, the remote Git repository is updated with the latest changes that you committed. Other team members can access and work with the most recent version of the codebase.

8. A *webhook* is a mechanism that allows real-time communication between different systems. In the context of Git and ARCAD Builder, you can set up a webhook to notify Builder about certain events in the remote repository. This integration is enabled by copying the GitHub webhook from Builder's webhook processing tool, which is known as "smee." To use this feature, ensure that webhook processing is activated in Builder, and use the provided webhook link (for example, <https://smee.io/IzfhozWff1rfG10t>) to establish the connection, as shown in Figure A-27.

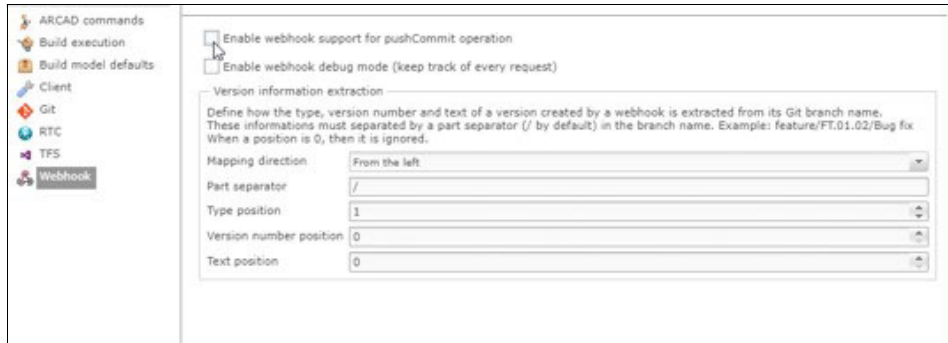


Figure A-27 Webhook integration between Git and ARCAD Builder

9. Run the `check version` command to generate an automatic commit in your local repository. By pulling from your local repository, you retrieve the latest commit message from the remote repository. This process helps ensure that you are always working with the most up-to-date code and information, promoting collaboration and reducing potential conflicts.
10. To incorporate the necessary IBM i views, right-click CHE, as shown in Figure A-28.

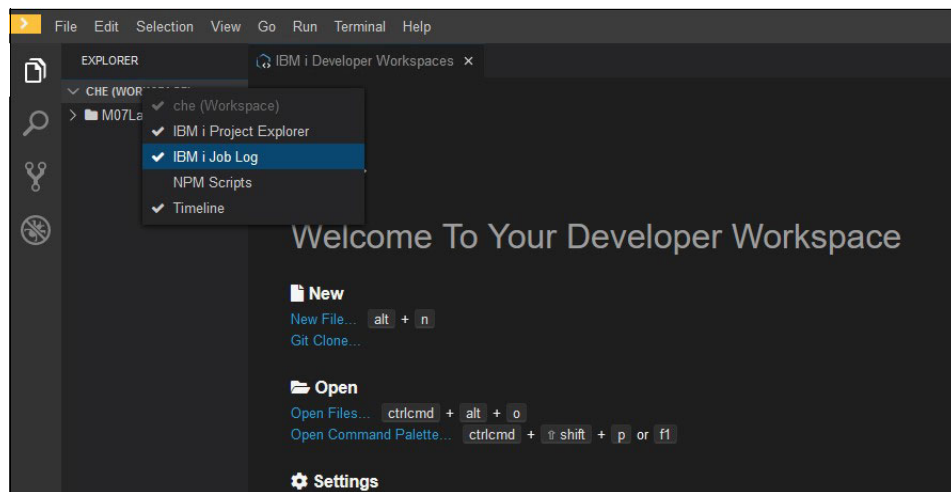


Figure A-28 IBM i views integration in CHE

## IBM i project explorer

IBM i project explorer offers essential features for integration and efficient development within the IBM i environment:

- ▶ Variables: Manage and track variables that are used in your development projects to help ensure accurate data handling and processing.
- ▶ Library lists: Configure library lists to access the necessary libraries and resources for your projects without performing a manual setup.
- ▶ Object libraries: Access and organize object libraries efficiently to simplify the management of your IBM i resources.
- ▶ My queries: Use built-in query functions to retrieve specific information from your IBM i system to enhance your ability to gather relevant data for your projects.

## ARCAD view

The ARCAD view is a UI that is provided by the ARCAD software suite. It offers a consolidated and organized perspective into various aspects of the software development lifecycle.

- ▶ Sites: Gain a comprehensive overview of your different development environments or locations so that you can organize and navigate your projects effectively.
- ▶ Builds: Track the progress of builds to help ensure a clear understanding of the status of your development efforts.
- ▶ Versions: Access and manage different versions of your projects. Each version is linked to a specific branch, so you can navigate between various stages of development.

## Prompting

*Prompting* in the context of IBM i Merlin refers to the interactive assistance that is provided to developers during various stages of application development. It offers guidance and suggestions as developers write code, which helps create accurate and efficient programs. Prompting enhances the development experience by reducing errors, improving consistency, and increasing productivity.

## Changed source

A changed source in the IBM i Merlin platform is tracked through a “Modified” flag, which indicates that alterations were made to the code. These changes are managed within the platform's source control system.

## Git compare

IBM i Merlin offers a Git compare feature that developers can use to efficiently analyze differences between various versions of source code. This tool enhances collaboration by providing an intuitive visual representation of changes, which helps with code review, error identification, and maintaining code quality throughout the development lifecycle.

## Git process

As part of the Git process within IBM i Merlin, developers can conveniently stage changes and commit them locally. This approach helps ensure that modifications are organized and tracked effectively before they are pushed to the shared repository, which contributes to a structured and controlled development workflow.

## Central Git repository

In IBM i Merlin, the central Git repository serves as a central hub for collaborative development. When developers push their changes to a specific branch in this repository, the repository is updated with the latest modifications, which promotes efficient collaboration and version control within the development team.

## Changes that are received by Git

When changes are committed and pushed to the central Git repository, these modifications are received by Git, which helps ensure that the latest updates are accessible to all team members that are collaborating on the project.

## ILE RPG offers extensive support

ILE RPG, with its extensive support, offers a range of features for enhanced development:

- ▶ **Outline:** Provides a clear overview of the code structure.
- ▶ **Model creation:** Simplifies the representation of the code's logic.
- ▶ **Simple navigation:** Streamlines moving between different sections of the code.
- ▶ **Hover information:** Displays contextual information when hovering over elements.
- ▶ **Procedure definition information:** Presents details from procedure definitions.
- ▶ **Collapsible code blocks:** Collapse sections of code for improved readability and focus.
- ▶ **Procedure call analysis:** Right-clicking procedures shows references and usages of them, which provides a deeper understanding of their impact.

## Tokenization

*Tokenization* in the context of IBM i Merlin refers to the process of categorizing and highlighting different elements in your code by using appropriate colors. This visual differentiation helps you quickly identify and distinguish between various components within your codebase, which can lead to improved readability and understanding.

## IBM i Merlin Code Formatting

IBM i Merlin Code Formatting helps ensure consistent and organized code for readability. Customize preferences for automatic formatting alignment with your style. Right-click selected code and choose **Reformat** to instantly apply the chosen rules. These rules maintain uniformity and enhance comprehension.

## IBM i Merlin Refactoring

IBM i Merlin Refactoring enhances code structure and readability without sacrificing functions. Rename symbols intelligently, and update code and models for consistency. Pressing Shift+Enter previews the changes before you commit them. The model auto-updates modifications. This approach helps ensure accurate and efficient code improvement.

## Content Assist

Content Assist in IBM i Merlin provides intelligent suggestions as you code. Press Ctrl+Space to start it. It works for both language and model, which helps enhance accuracy and speed. The live problem view identifies and highlights issues, and you can do direct navigation and automatic updates as you patch them.



## SQL

SQL in IBM i Merlin brings advanced features for efficient database interaction and management:

- ▶ **Tokenization:** Clearly divides SQL into meaningful elements for comprehension and editing.
- ▶ **Formatting:** Helps ensure consistent and readable SQL code by automatically applying formatting rules.
- ▶ **Code collapse:** Organizes SQL blocks to make it simpler to navigate and focus on relevant sections.
- ▶ **Embedded SQL:** Integrate SQL statements within host languages to enhance database interaction within the application code.

## ARCAD Transformer RPG

ARCAD Transformer RPG is a powerful tool that facilitates the modernization of RPG code to enable it to adapt to contemporary coding standards and practices. However, there are certain aspects that the transformation process does not consider:

- ▶ **Specifications that are not available in Free Form:** Traditional I and O specifications are not available in Free Form RPG.
- ▶ **Not managed in F / D specs:** Certain elements such as primary files (P), secondary files (S), table files (T), or address files are not managed in Free Form RPG. Also, D-specs with FROMFILE / TOFILE clauses are excluded.
- ▶ **Unconverted operations:** Certain RPG operations such as MHHZO, MHLZO, MLHZO, MLLZO are not automatically converted during the transformation process.

Furthermore, there are specific cases where operation codes cannot be converted:

- ▶ **TIME:** If the result field length is equal to 14 characters.
- ▶ **SCAN, CHECK, CHECKR:** When the result field is an array.
- ▶ **BITON, BITOFF:** When factor 2 is a named constant.
- ▶ **POST:** When the result field (data structure name) is used.
- ▶ **MOVE, MOVEL:** When the factor 2/result is a varying-length field.
- ▶ **MOVEA:** When the field is defined as a CONST parameter.
- ▶ **KLIST, KFLD:** When located or used in a COPY clause.
- ▶ **CALL, PARM:** For CALL operations, the indicator "LR" (positions 75-76) and the CALL Pgm(idx) syntax are not converted.
- ▶ **GOTO, TAG:** GOTO within a subprocedure and TAG in the "Main" program, or when GOTO and TAG do not comply with structured programming. Also, when TAG is used by WHENEVER GOTO (SQL).

## IBM i Merlin requirements

The installation of IBM i Merlin is flexible. You can install it by using either the Red Hat OpenShift web console or the CLI (the `oc` command). This versatility provides options for a convenient installation process that is tailored to your preferred method.

### Versions of the product that are compatible and officially endorsed

Table A-4 provides a comprehensive overview of the versions that are both compatible and officially endorsed for the installation of IBM i Merlin. The following table presents a clear snapshot of the supported versions so that you can make informed decisions that pave the way for a successful implementation of IBM i Merlin within your IT environment.

Table A-4 Supported versions of IBM i and Red Hat OpenShift Container Platform for IBM i Merlin

IBM i	Red Hat OpenShift Container Platform
7.3	4.8
7.4	4.9
7.5	4.10

### Necessary Red Hat OpenShift resources

Table A-5 outlines the crucial CPU requests, CPU limits, memory requests, and memory limits that play a pivotal role in helping ensure the optimal functioning of IBM i Merlin within the Red Hat OpenShift environment. This adherence to resource allocation guidelines is integral to the successful implementation of the IBM i Modernization Engine.

Table A-5 Necessary Red Hat OpenShift resources

Name	CPU request	CPU limit	Memory request	Memory limit	Note
IBM i Merlin	2.5 <sup>a</sup>	5 <sup>b</sup>	7G <sup>a</sup>	15G <sup>b</sup>	None.
IBM i Developer Tool	0.5 <sup>a</sup>	2.7 <sup>b</sup>	1.5G <sup>a</sup>	3G <sup>b</sup>	The resource is per each instance. <sup>c</sup>
IBM i CI/CD	0.5 <sup>a</sup>	1 <sup>b</sup>	1G <sup>a</sup>	2G <sup>b</sup>	The resource is per each instance. <sup>c</sup>

a. Request signifies the minimum required amount.

b. Limit signifies the maximum anticipated utilization.

c. When an administrator installs either of IBM i Developer or IBM i CI/CD Tools within an Red Hat OpenShift project, it corresponds to one instance.

## IBM i requirements

Here are the prerequisites for the IBM i environment:

- ▶ IBM i 7.3 or later, complemented by the latest application of the HTTP PTF Group.
- ▶ IBM Rational Development Studio (5770-WDS) is an essential requirement for the compilers, enabling the conversion of source code into object code.

## Entitlement

Clients can acquire IBM i Merlin through [IBM Passport Advantage](#). After you purchase IBM i Merlin, authenticate by using your IBMid on Passport Advantage. A designated entitlement key that is associated with the acquired product is activated within IBM Marketplace. This entitlement key coupled with an active paid entitlement grants you access to the container images that are available within the Entitled Registry.

**Price:** IBM i Merlin follows a “per-developer” pricing model, aligning with its deployment within the Red Hat OpenShift Container Platform. By using the inherent license monitoring mechanism of Red Hat OpenShift Container Platform, IBM i Merlin employs the Virtual Processor Core (VPC) framework. To secure your IBM i Merlin entitlement, place an order for one VPC unit per developer, which creates an individual CodeReady workspace for each developer. This offering is available at a rate of \$4500.00 per VPC.

### **Installing IBM i Merlin in an air-gapped environment**

For more information about installing IBM i Merlin in an air-gapped environment, see [Install IBM i Modernization Engine for Lifecycle Integration in AirGap environment](#). This resource covers essential prerequisites, the setup of a Bastion host, configuration of the local Docker registry, installation procedures for IBM i Merlin, and comprehensive guidance about mirroring images and configuring the cluster. Also, you can find clear instructions for creating the IBM i Merlin catalog source.

For more information about installation IBM i Merlin and more, see [GitHub](#).



# Abbreviations and acronyms

<b>ACFS</b>	ASM Cluster File System	<b>HDBLCM</b>	SAP HANA Database Lifecycle Manager
<b>ACS</b>	Access Client Solutions	<b>HIPAA</b>	Health Insurance Portability and Accountability Act
<b>AI</b>	artificial intelligence	<b>HMC</b>	Hardware Management Console
<b>API</b>	application programming interface	<b>laaS</b>	Infrastructure as a Service
<b>ASM</b>	Automatic Storage Management	<b>laC</b>	Infrastructure as Code
<b>BCDR</b>	business continuity and disaster recovery	<b>IASP</b>	independent auxiliary storage pool
<b>BOB</b>	Build on Build	<b>IAVA</b>	Information Assurance Vulnerability Alerts
<b>BSL</b>	business source license	<b>IDE</b>	integrated development environment
<b>C2S</b>	commercial cloud service	<b>IFS</b>	Integrated File System
<b>CD</b>	continuous deployment	<b>ILE</b>	Integrated Language Environment
<b>CDB</b>	Container Database	<b>IoT</b>	Internet of Things
<b>CI</b>	continuous integration	<b>IPL</b>	initial program load
<b>CI/CD</b>	continuous integration and continuous deployment	<b>ISA</b>	Instruction Set Architecture
<b>CIS</b>	Center for Internet Security	<b>ISV</b>	independent software vendor
<b>CJIS</b>	Criminal Justice Information Services	<b>JFS</b>	Journaled File System
<b>COLO</b>	Colocation	<b>LPAR</b>	logical partition
<b>CRN</b>	Cloud Resource Name	<b>LPM</b>	Live Partition Mobility
<b>CVE</b>	Common Vulnerabilities and Exposure	<b>LPP</b>	Licensed Program Product
<b>DBA</b>	database administrator	<b>LVM</b>	logical volume manager
<b>DCM</b>	dual-chip module	<b>Merlin</b>	Modernization Engine for Lifecycle Integration
<b>DISA</b>	Defense Information Systems Agency	<b>MGMTDB</b>	Management Database
<b>DLPAR</b>	dynamic LPAR	<b>MMA</b>	Matrix Math Accelerator
<b>DoD</b>	Department of Defense	<b>MPL</b>	Mozilla Public License
<b>DR</b>	disaster recovery	<b>MSP</b>	Managed Service Provider
<b>DSL</b>	domain-specific language	<b>NAS</b>	Network Authentication Service
<b>EDA</b>	event-driven automation	<b>NIM</b>	Network Installation Manager
<b>EIM</b>	Enterprise Identity Mapping	<b>OCR</b>	Oracle Cluster Registry
<b>FQCN</b>	fully qualified collection name	<b>OMI</b>	Open Memory Interface
<b>GDPR</b>	General Data Protection Regulation	<b>OS</b>	operating system
<b>GIMR</b>	Grid Infrastructure Management Recovery	<b>OSPP</b>	Operating System Protection Profile
<b>GTS</b>	Global Technologies Services	<b>OTN</b>	Oracle Technology Network
<b>HA</b>	high availability	<b>PAS</b>	Primary Application Server
<b>HADR</b>	high availability and disaster recovery	<b>PASE</b>	Portable Application Solutions Environment
<b>HCL</b>	HashiCorp Configuration Language	<b>PGM</b>	program
		<b>PTF</b>	Program Temporary Patch

<b>QA</b>	quality assurance
<b>RAC</b>	Real Application Clusters
<b>RAS</b>	reliability, availability, and serviceability
<b>RBAC</b>	role-based access control
<b>RCA</b>	root cause analysis
<b>REST</b>	Representational State Transfer
<b>RFC</b>	Remote Function Call
<b>RHEL</b>	Red Hat Enterprise Linux
<b>RPG</b>	Report Program Generator
<b>SaaS</b>	Software as a Service
<b>SAN</b>	storage area network
<b>SCM</b>	source code management
<b>SCN</b>	SAP Company Number
<b>SEU</b>	Source Entry Utility
<b>SLIC</b>	System Licensed Internal Code
<b>SME</b>	subject matter expert
<b>SSH</b>	Secure Shell
<b>SSO</b>	single sign-on
<b>SSP</b>	Share Storage Pool
<b>STIG</b>	Security Technical Implementation Guide
<b>SWPM</b>	Software Provisioning Manager
<b>VCS</b>	version control system
<b>VIOS</b>	Virtual I/O Server
<b>VM</b>	virtual machine
<b>VPC</b>	Virtual Processor Core
<b>VS1</b>	Virtual Server Instance
<b>WSL</b>	Windows Subsystem for Linux
<b>YUM</b>	Yellowdog Updater, Modified

# Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this book.

## IBM Redbooks

The following IBM Redbooks publications provide more information about the topics in this document. Some publications that are referenced in this list might be available in softcopy only.

- ▶ *Deploying SAP Software in Red Hat OpenShift on IBM Power Systems*, REDP-5619
- ▶ *IBM Power Systems Cloud Security Guide: Protect IT Infrastructure In All Layers*, REDP-5659
- ▶ *Introduction to IBM PowerVM*, SG24-8535
- ▶ *Oracle on IBM Power Systems*, SG24-8485

You can search for, view, download, or order these documents and other Redbooks, Redpapers, web docs, drafts, and additional materials, at the following website:

[ibm.com/redbooks](https://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](https://ibm.com/support)

IBM Global Services

[ibm.com/services](https://ibm.com/services)





**Redbooks**

**Using Ansible for Automation in IBM Power Environments**

SG24-8551-00

ISBN 0738461873



(0.5" spine)

0.475" x 0.873"

250 <-> 459 pages







SG24-8551-00

ISBN 0738461873

Printed in U.S.A.

Get connected

